Release 12 Java Infrastructure

Brian Bent
Solution Beacon, LLC
Lance Reedy
IBM

Abstract

In this presentation, learn about the new Java infrastructure underlying the Release 12 environment. The new technology delivers significant changes to how the application server works. Topics covered include the architecture of the new technology components and the administration tasks necessary to support them.

Objectives

- Describe the Java Infrastructure underlying the Release 12 environment
- Describe the technology components and administration tasks necessary to support them
- Describe tuning considerations for managing the Java infrastructure

Introduction

This whitepaper will focus on three aspects of the Java infrastructure in Release 12. First is the one that our end users see most often, the use of Java in the web browser. We'll see that the changes enable us to remove one more custom component from the environment which is always a good thing! Second is the heart of the technology stack: the applications server. The new Applications Server 10g is built on top of an entirely new Java engine, which we'll call OC4J for the moment. Finally, we'll look at some of the utilities and knowledge needed to support the new Java infrastructure.

Java in the Web Browser

One of the most talked about changes in Release 12 is the elimination of JInitiator. As a quick review, JInitiator was Oracle's own licensed distribution of Sun's Java browser plugin for running applets inside the browser, and was required for using the forms based applications in the Release 11*i* E-Business Suite. In an effort to more seamlessly integrate into the corporate infrastructure Oracle is moving away from using a custom patched browser plugin. For the initial Release 12 environment Oracle did require a custom patched version of a recent Sun browser plugin, based on the Sun's 1.5.0_10 Java distribution. This browser plugin, while more modern than that used in previous Release 11*i* implementations, is nonetheless a non-standard component. The new browser plugin can be identified as being Oracle specific by the –erdist suffix on its name.

For users of Internet Explorer this custom version of the Java plugin should download and install upon initial connection to the applications server due to an ActiveX control being used. For users of other browsers that don't support Active X, such as Firefox, a simple but nonetheless manual step is required. MetaLink Doc. ID: 393931.1 states Oracle's goal is to support all newer releases of the java runtime, and on June 25th 2007 they moved forward with this by announcing support for use of Sun's unpatched 1.5.0_12 JRE for not only release 12 but also for implementations of Release 11*i* that meet the patch requirements.

So the question is what do we gain by moving away from JInitiator? Why the big deal? The answer is a couple of things really. For one thing we get to remove a custom component in the environment. For another it simplifies the client environment to help reduce support issues. Finally it gives end-users increased access to technology improvements directly from Sun; no more waiting for Oracle to roll them into their release of JInitiator. For instance there is a project underway at Sun to reduce the startup time of the JVM, with emphasis on the client environment such as we're using here. When this project goes into production in the 1.6 Java release it should provide a direct benefit to our users.

There are a variety of MetaLink notes that are relevant to this discussion. For new Release 12 implementations you should review Doc. ID: 393931.1 detailing the process for installing Release 12's initial 1.5.0_10-erdist Java environment on the client. A second is Doc. ID: 389422.1, which details installing the initial Java environment and other settings needed to use browsers other than Internet Explorer, most notably Firefox. And finally there's Doc. ID: 290807.1 on what is needed to upgrade a Release 11*i* environment to use 1.5.0_12+ or 1.6.0_03+ Java environment as provided by Sun.

Application Server 10*g*

The first thing you need to know about the server side of Release 12 is that it's built out of a new apps server, Application Server 10g. In fact AS 10g is built out of two versions of it; there's the 10.1.2 version used for concurrent processing and the 10.1.3 version used for the remainder of the applications suite. Oracle Containers for Java, also known as OC4J, is the application server component at the heart of the AS 10g package. It is a direct replacement for JServ, and works via a connection to Apache, just as JServ did. OC4J supports the same multi-tier deployments that were possible in the past. So, the only downside here is that Apache does not receive an upgrade, although the patch level changes. Oracle stays with the aging 1.3 version of Apache. On the bright side it means that existing skills configuring 1.3 don't have to be thrown away, since an upgrade to 2.0 would bring a whole new configuration file syntax. That can be important if you're using advanced configurations of Apache such as URL firewalling, or URL rewriting. It also means that we know where things are likely to break.

Having two versions of the apps server means having two ORACLE_HOMEs to manage…two sets of binaries to patch, and two sets of config and logfiles to work with. The 10.1.2 environment runs the concurrent manager server, and uses a version 1.4 JVM in its initial installed configuration. The 10.1.3 environment runs the E-Business Suite apps themselves. By default it runs in a version 1.5 Java virtual machine, and supports newer Java Enterprise Edition 1.4 standards. This means that it includes many new features that interest developers if you're doing custom development. If you're merely developing customizations to the E-Business Suite you'll be restricted to using the same core features that the Suite uses, but you still might find some value in the new capabilities.

For those not familiar with Sun's marketing names, Java 5 equates to the version 1.5 Java environment. Use of version 1.5 of Java brings us some improved performance and new support for management utilities. The Java EE refers to Sun's Java Enterprise Edition standards, which are a series of APIs on top of the core Java language, these form the basis for their own ecosystem in the Java marketplace, with Oracle, IBM, and BEA all offering their own applications servers. If you're interested in more information about the changes in the environment, and haven't been there before, check out Steven Chan's blog on Oracle's blog site. He is a Director at Oracle and his blog regularly deals with issues concerning the E-Business Suite, including issues like upgrades, support tips, and integration.

Important features in 10g can be broken into two categories: important Oracle components bundled in AS 10g, and industry standards in it. This list is by no means comprehensive, there are certainly other components bundled into AS 10g, these were just selected because of their relevance to Release 12. On the Oracle side of the features, we have the Business Process Execution Language (BPEL) process manager, which allows business logic to be implemented in the form of a modeled language; this separates the business logic from the code that implements it, making the environment more agile. It also forms part of the glue needed for things like web services. Additional plumbing for web services are present in the form of the Web Services manager and Enterprise Service Bus, which provide the infrastructure and a transport mechanism. Together these three components form the basis of much of the web services infrastructure. Finally there's Oracle's Identity Management, which integrates the environment into security platforms such as LDAP. This component often drives standalone implementations of AS 10g as part of an enterprise security platform to integrate the E-Business Suite into LDAP or Active Directory.

Continuing on to the industry standard side of the house, there is support for the Java Enterprise Edition 1.4 standards. Sun's Java Enterprise Edition adds a large collection of APIs and toolkits to the Java language for scalable, flexible applications. The 1.4 version brings recent innovations like Enterprise Java Beans 3 which use an annotation based model for moving data between the database tables and live Java objects. Many of these new features have yet to be implemented in the Release 12 codebase, but they are important to understand as many of them will be appearing in the forthcoming Fusion releases.

If you're doing any custom development in the environment, or are likely to be an advanced implementation of Fusion, then review http://www.oracle.com/technology/tech/java/oc4j/1013/OracleAS-NF-10131.pdf to learn a little about some of the new technology features that are available to you. With each new release of the apps server, Oracle is moving further into the Java Enterprise Edition market, adding more and more APIs, which brings new features and programming models to learn about. If you're doing development in the E-Business Suite then you'll be limited to the portions of the APIs that Oracle is using right now. But if you're writing a custom application from scratch, then you're able to use a much broader swath of the language. It's also worth noting that Oracle has stated that they are moving to a much larger set of the Java EE APIs when they write the Fusion applications. So even pure E-Business Suite development will eventually take you into these new technologies.

Let's look at the OC4J technology to learn a little bit about how AS 10g operates. Our goal is to understand enough of the technology to know what moving parts are present, and which ones we're likely to have to work with as an Apps DBA. First is the traditional front end of Apache, which performs all communications with the end-users. Apache also offers us optional features such as URL firewalling which can mask security sensitive URLs such as administrative pages from an Internet facing application. OC4J itself runs in the form of three distinct servers, each being its own independent Java process. The oacore instance supports framework based web applications. The forms instance supports Oracle forms based applications as the name says, and finally oafm runs any web services that may be present in the environment. There's a fourth instance which can optionally be setup that runs the Application Server Control application, which is part of OEM and provides an administrative web interface for the environment. In the E-Business Suite using AS Control is not necessary, as the traditional Applications DBA (AD) utilities support what is needed to manage the server components. But if you're doing custom application development, then it might provide some capabilities that you'd be interested in, like the ability to deploy and control your custom application independently of the rest of the E-Business Suite. Another part of the environment is the Oracle Process Monitor, or OPMN. This component is used for stopping and starting OC4J and the http_server, in other words Apache, as well as the various OC4J servers. In an E-Business Suite implementation the application server is not controlled directly via OPMN, but is instead controlled via the traditional AD utilities such as adopmnctl.sh.

With Release 12, implementations will remain a N-tier architecture, meaning that tiers can be distributed or combined to meet the business needs. In all configurations there will be three logical tiers, the Web Tier running the Apache server and its connection to OC4J, the Application tier running the OC4J apps servers, and finally the database tier running the Oracle Database 10g. These three logical tiers exist in all AS 10g deployments, and it is entirely possible to run all three tiers on a single server or on three separate servers. To mitigate the security risks of running everything on one server, the environment can also be split into separate servers, and firewalls can be placed between every tier. For purely internal deployments the question of one server versus several primarily comes down to scalability, with security concerns a distant second. For an Internet facing deployment splitting the environment into separate servers is a necessity. By definition the highest risk component is the one connected to the Internet, so this would be the web tier, which would be isolated from the other tiers in every possible way. Additionally, to help improve scalability and availability of the applications, clustering can be used at each tier. At the web tier, clustering would involve multiple servers each running their own copies of Apache and mod_oc4j. Network devices typically called content switches would be used to route clients between the different nodes in the web tier, and to route around node failures. Content switches are separate network components from vendors like Cisco or F5 Networks. At the applications tier, multiple servers would each have all 3 OC4J instances running, and the web tier would handle directing client requests to them. And finally at the database tier Oracle Real Application Clusters could be used to spread one physical database across multiple servers. Review MetaLink Doc. ID: 380490.1 concerning Release 12 Configurations in a DMZ.

The Java 5 improvements apply not only to the server components, but also to the client now that JInitiator is being replaced with a Java Runtime direct from Sun. The most wide ranging benefits will be to those who are doing custom development, by having access to the Java 1.5 language you'll be able to make use of some newer language features such as Generics, AutoBoxing, and Annotations all of are intended to simplify development and/or increase reliability when the application is running. An incremental improvement present in Release 12 is the removal of the need for a graphics device to support the applications. The 1.5 Java language no longer needs something like the system console, or the Xvfb or VNC server to support drawing graphics such as charts, thus removing one more component from the environment.

There are several types of performance improvements that are present in the 1.5 Java technology itself, and therefore inherited by the Oracle applications. First is improved memory management techniques which should result in better OC4J server performance, particularly during periods of heavy workload. Additionally enhancements to the Java virtual machine allow it to make better use of modern processors and their 64 bit instruction sets. This brings a direct increase in performance for applications such as the E-Business Suite. The final performance improvement is that overall memory usage has been improved through features like shared memory, where portions of the JVM's memory footprint can be shared between multiple JVMs, thus eliminating duplication, and improving launch time for subsequent JVM processes. The other side of this is that as memory usage in the JVM goes down Oracle is moving more code into the JVM, so you can expect the memory footprint of the JVM to go up over time.

Management Utilities

In terms of tools for managing the environment there are three entities we'll focus on in this paper: the jps command which is equivalent to the UNIX ps command and displays a list of running java processes, and all of their options. Jstat complements the UNIX vmstat or iostat commands, and shows details of resource usage within a running JVM. And finally the JConsole application is a GUI tool for looking at what's happening within a running java application. It's a very good tool for helping tune the Java environment. It's important to note that these utilities are not included in the Oracle installed Java environment on the server. The Sun's Java developer's toolkit must be installed outside of the AS 10g ORACLE_HOME, which typically installs somewhere like /usr/java.

For those that aren't fond of UNIX tools like grep, jps can be used to display details about all of the java processes being run by your userid. When invoking the jps utility with a –v option will tell you not only the processes, but all of their command line arguments as well, which is helpful for telling the three instances of OC4J apart. It will also show us the current memory configuration of the JVM, which is a prerequisite for being able to tune the JVM configuration.

Jstat complements the UNIX vmstat and iostat utilities, and is quite useful when writing a shell script for automatically monitoring the JVM performance and/or availability. The command uses the UNIX process id to identify which JVM it will report on, and an interval to pause between displays, which is similar to the vmstat and iostat commands. Passing jstat a –gccapacity parameter will show the utilization and the maximum allowed size of some of the JVM memory pools. This is the common use for jstat when writing a monitoring script. For a real reference, check the man page for jstat, or Sun's online documentation, to see which options will show you the numbers that you want to monitor.

The third utility relevant to us is JConsole, a GUI tool for interactively exploring a running JVM, and for monitoring the most common tunable entities such as the memory pools. It's worth noting that Java has several different memory pools, and while we cannot teach you what tuning you'll need to do, we can show you the mechanism that you'll need to use. The tuning that you'll need to do will vary wildly based upon the number of users, and applications in use in your implementation. One key concept to know is that by default JConsole will not connect to our OC4J JVM's. An additional parameter -Dcom.sun.management.jmxremote will need to be passed to each JVM in order to enable connections by JConsole. Review Doc. ID: 416455.1 for more information on the jmxremote parameter in Release 11*i* and Release 12.

JConsole will present a simple graphical user interface that can be used to see current memory usage of the JVM, which is a prerequisite for knowing when you'll need to tune it. Exhausting any one of the JVM memory pools, having it hit 100% utilization, will cause the JVM to shutdown immediately. Note that for the Heap pools the JVM's memory management utility (the Garbage Collector) will periodically collect free memory. It is entirely possible that there can be so many active users in the applications that no memory can actually be freed, resulting in the Heap memory filling up and the JVM shutting down. While JConsole can help us watch the JVM during an activity, such as a particular test scenario, it's only really useful when it has a pair of eyes looking at it. This is why the jstat command will provide us with these same metrics, albeit in a harder to use format. The OC4J logfiles and jstat information written to a logfile will be useful when trying to debug a crash that has occurred off hours when you're not actively watching JConsole.

Administering AS 10*g*

Strictly speaking the AS 10g processes should be controlled via the AD utilities, and those are discussed in detail in Oracle's documentation. But to understand how we'll be tuning the JVM let's take a moment to consider what the AD utilities are themselves doing. The Oracle Process Monitor is used to launch, monitor, and stop the OC4J and Oracle HTTP Server processes. In that light it has configuration data that tells it how to launch the OC4Jj instances, and what parameters to pass to them. This configuration is what we'll have to modify using the preferred methods of the E-Business Suite. To properly tune OC4J you'll use a combination of three things: The Oracle Context file to set the memory parameters passed to the JVM at launch. Then you'll use AutoConfig to apply those parameters to OPMN's configuration file. Finally you'll use the AD commands as a front-end to stop and start OC4J.

The command used by OPMN to actually control the environment is, quite appropriately, opmnctl. It's located in a new directory structure that has appeared in Release 12, the INST_TOP. This structure contains all of the configuration and log files for an Oracle instance. The idea is that there are really two types of files in an Oracle instance: those that are unique to the instance and change at installation or run time, and those that are not unique such as binaries and resource files. So Oracle has split out the unique files to the INST_TOP in order to make it easier to share the non-unique content between multiple installations using something like a NFS for instance.

Tuning the Java virtual machine is typically done when it's launched, and OC4J is no exception. In the opmn.xml file, there is a stanza for each of the OC4J instances, manipulate this stanza via the Context file in order to tune the JVM's memory usage. Modifing the opmn.xml directly will result in your changes being undone the next time AutoConfig is run, which of course is not the desired result. You can identify which instance of OC4J in the stanza identified by the "program-type id=" entry. The parameters that are passed to the JVM are located in the "start-parameters" section of the stanza. So we now know where to tune OC4J, but we're not much closer to knowing what to tune. Unfortunately tuning the JVM is not an exact science. It depends upon a number of variables like how many users are present, what applications are being using, how much data is present, and so on. At this time there are no real ballpark recommendations that can be made, such as amount of memory per user, but fortunately there are only a couple of core values that typically need adjustment.

There are a variety of command line options that can be tuned in order to control the JVMs. Note that these are all very sensitive to capitalization and use of spaces, so be sure to type them accurately. Most memory parameters in the JVM take an initial value which is used to size the pool when the JVM starts, and a max value which represents the ceiling that the pool can grow to. Note that these are all specified in megabytes. First is the initial size of the Heap memory (-Xms), and next is the max size of the heap (-Xmx). Adjusting the max size is the one tuning change that will be made most frequently. Adjusting the initial size isn't really that important, focus on the max size. Next is another memory pool that problems have been identified with, called the Permanent Generation. It holds entities that will be present for the life of the JVM. Things like the skeleton code for every class type that has been loaded. The parameters to tune the Permanent Generation pool are -XX:PermSize and -XX:MaxPermSize. Garbage Collection does little if anything to free up memory in this pool. So should you see its capacity approaching its maximum size make a note that you'll need to shutdown and increase the max size soon! And finally there is the parameter -Xloggc that details how much time is spent doing garbage collection and logging it to a file for later analysis.

To summarize the process for making tuning changes, start by making your changes to the Context file. While changing the opmn.xml directly can be useful for a quick and dirty experiment on a development server, be warned that the next AutoConfig run will remove your changes! The second step is to run AutoConfig to apply the changes to the active opmn.xml file. And next you must restart the OC4J instances using the AD utilities to make the changes take effect. Be sure to use tools like jps to check the running instances of oc4j afterwards to make sure that your changes have actually taken effect!

The most common way to know an OC4J instance needs to be tuned could be that it hits an out of memory exception and shuts itself down! So to be able to track these occurrences, find the OC4J logfiles. Because these are per instance files, they are located under the INST_TOP! Under the INST_TOP is the logs/ora/10.1.3 directory tree, and yes, there is a separate logs/ora/10.1.2 tree for the concurrent manager server. The Apache logs are conveniently located in the Apache directory, and the OC4J logs located in the somewhat oddly named j2ee directory tree.

There are several logfiles that are extremely useful in troubleshooting. The Apache/access_log files will contain a list of all URLs that have been accessed by the end-user and how long they request took to process. The error_log file will contain requests that failed to complete because of things like invalid file names. These result in HTTP 404 errors as seen on the client web browser. For Internet facing applications, review this file periodically to look for activity by hackers and Internet worms. These type of security breaches try to retrieve resources such as administrative pages, or they try to pass very large amounts of data to pages in hopes of causing a buffer overflow. It is a best practice to have a security group or application review these logs looking for this type of suspicious activity. If URL rewriting is in use, the mod_rewrite.log will contain an audit trail of these activities. URL rewriting is used in some configurations to mask the real name of the applications server. There could be situations where the end-users do not need to know the actual application server name, and instead configure the application so it is accessed using a virtual name such as sales.mycompany.com. Finally the OC4J servers write logs to a per instance directory structure in the logfile application.log. Messages such as the out of memory error that would require entering the tuning exercises listed in this paper.

In summary, the Release 12 environment introduces several important java technology changes: a new client component that should work better in a real-world corporate setting and most importantly a new application server which will change our administration procedures as compared to Release 11*i*. These changes bring many new capabilities to the applications. This infrastructure technology change also gives us a sneak peak of the technology that will be behind the Fusion applications when they debut. What Oracle is really doing is maturing some of the Fusion Middleware out in the field so that we can kick the tires and see how it works. Release 12 also leaves us with some old components, namely Apache 1.3. Next is the increase in Java Enterprise Edition compliance, which is important for several reasons. One is development in this environment is becoming less of a specialized skillset. The other aspect is it makes our application deployments less specialized and makes integration into 3rd party vendor applications easier. And finally we get some long overdue tools for helping to monitor and manage the Java components running on the applications tier.