IBM WebSphere Compute Grid and WebSphere Java Batch in WAS V8.5

Getting Started with Java Batch

See the WP101783 White Paper for more details

Introduction

A common question that comes up is this:

"I'm interested in WebSphere Java Batch. How do I start? What are some common approaches to begin working with this solution?"

It should be no surprise the answer is *not* to rip-and-replace everything you have working today. Indeed, the approach patterns are all about *incremental adoption*. The objective is to *minimize* processing disruptions and move to WebSphere Java Batch in a controlled manner.

This document will provide an overview of several proven approaches for beginning the use of WebSphere Java Batch[1].

Any Platform

WebSphere Java Batch runs on many different operating system platforms[2]. The first several approach patterns spelled out in this document apply to *any supported platform;* the final two focus on z/OS patterns where z/OS-exclusive features involved.

Overview

The following list provides a high-level view of the patterns discussed in this document:

Develop net-new batch in Java and WebSphere Java Batch

If new batch processes are called for, make the decision to develop new processes in Java. Leave existing non-Java batch as is and refocus new batch development on Java. Integrate non-Java and Java into larger batch processes with your enterprise scheduling software.

Reuse existing Java "main" programs

If your environment has an inventory of Java batch programs written as Java "main" programs, those may be easily packaged (or "wrappered") in a WebSphere Java Batch program pattern to allow execution in the WebSphere Java Batch runtime.

Re-architect existing non-Java into Java batch

You may have existing non-Java batch programs you wish to re-architect for some reason, such as new business requirements imposed on the batch design. The decision to re-architect provides an opportunity to change the programming model to take advantage of WebSphere Java Batch.

z/OS: Take advantage of System z Specialty Engines

On z/OS you may wish to rewrite existing non-Java into Java to take advantage of offload to System z specialty engines (zAAP, or zAAP-on-zIIP). The objective is not so much to re-architect (though some of that may take place), but to simply *replace* non-Java with Java.

z/OS: Call existing COBOL from new Java batch programs

If you have COBOL modules that provide useful batch services, you may choose to leverage those within the context of a Java batch program. WebSphere Java Batch on z/OS provides a "COBOL Container" function to do just that.

And of course combinations of these approaches may be considered when planning your approach to using WebSphere Java Batch.

1 Another term commonly used is "Compute Grid," which is a reference to the IBM program product by that name. Compute Grid is still available, with the function also included with the WebSphere Application Server V8.5 product as well.

2 Windows, AIX, Linux, Linux for System z, HP-UX, Solaris, IBM i, and z/OS.

Note about Enterprise Scheduler Integration

A common question that comes up is how to integrate non-Java with Java in broader batch processing flows. The answer to that is to use a supplied WebSphere Java Batch function that allows integration with existing enterprise scheduler functions. That function is often referred to as *WSGRID,* which is the name of the supplied utility that performs the linkage between the enterprise scheduler and WebSphere Java Batch.

WSGRID works with any enterprise scheduler. If the scheduler is capable of issuing a command, invoking a shell script or submitting JCL, it can use WSGRID to integrate batch processes with Java batch running in WebSphere Java Batch[3].

Develop New Batch in Java

This approach provides a clean dividing line between existing non-Java batch processes (which continue to serve the business well) and net-new batch processes written in Java.

An advantage of this approach is it represents a minimal risk of disruption to *existing* batch processing. Existing batch is left unaffected; this approach addresses *new* batch requirements. This affords you a good opportunity to prototype and test the new Java batch routines before putting them into production.

Reuse Existing Java Main Batch Programs

In many environments the journey with Java batch started with simple Java programs written to be invoked by launching a JVM and running the program. These are sometimes referred to as "Java main" programs, or "legacy Java batch."

A common motivation to consider WebSphere Java Batch is the concern about how to manage a growing environment of Java "main" programs. These programs tend not to have much management structure around them. Beyond a small handful of such programs, concerns about management and support become a very real and pressing issue.

It is not necessary to discard or rewrite these programs when moving to WebSphere Java Batch. WebSphere Java Batch has a program pattern that allows these Java "main" programs to be wrappered with code that allows them to run within a WebSphere Java Batch managed environment. That provides a rapid path to reuse for those existing assets.

While this approach allows those assets to be quickly used in a WebSphere Java Batch environment, the approach does not provide much in the way exercising the rich set of features provided by the WebSphere Java Batch runtime[4].

A good practice is to plan time to rewrite these wrappered programs into Java batch applications that better leverage the WebSphere Java Batch runtime. The wrapper approach certainly works, but it leaves the full power of WebSphere Java Batch under-utilized.

3 WSGRID provides a message-based linkage between WebSphere Java Batch and the enterprise scheduler. The scheduler invokes the supplied WSGRID utility program, which sends a Java batch submit message to WebSphere Java Batch. The instance of WSGRID then stays active for the life of the Java batch execution in WebSphere Java Batch. Java Batch output streamed back over message interface.

4 Such as transaction management, checkpoint services, the Batch Data Stream framework for data read and write stream handling, and many others.

Re-architect Existing non-Java Batch

A move to Java Batch may well afford an opportunity to revisit the underlying architecture of the existing batch. It may be that new business requirements call for an update to the batch processing, and this may provide a good justification to consider re-architecting the batch process.

The *desire* to re-architect existing processes is not uncommon. Often what is needed is a good *catalyst* for making the change. The decision to adopt WebSphere Java Batch may provide the needed spark to undertake the architectural review. WebSphere Java Batch does *not* mandate it; it simply helps make the decision.

Note that the decision to re-architect existing non-Java does *not* mean all your non-Java batch needs review. Indeed, you may well have a very small subset of existing non-Java batch in mind for such review and update. That is also very common: where re-architecting is considered, it is often quite carefully limited to a few batch processes when starting out. In time other processes then come under review.

z/OS: Take advantage of Specialty Engines

The approach here is to identify existing non-Java batch processes and simply rewrite them to do the same thing in Java running in WebSphere Java Batch. The desire is to leverage System z specialty engines (zAAP or the newer zAAP-on-zIIP), which enable a cost advantage by offloading Java workloads from the general processor.

The key question here is what existing non-Java batch should serve as candidates rewrite. The primary considerations early in this process are: (a) reasonably limited scope, and (b) risk containment.

Limiting the scope makes simpler and more manageable the rewrite project. An effort to rewrite a company's *entire* batch process would likely fail under the weight of the sheer complexity of it. Better to identify a *smaller subset* to begin with. Better still if this subset is one with limited inter-dependencies with other parts of the overall batch process. Over time more and more batch processes become converted to WebSphere Java Batch.

Risk containment is also an important criteria for early rewrite candidates. Batch processing often has components that are part of the processing *critical path*[5], and other components that are supporting but peripheral to the core. So initially the focus is often on components with limited inter-dependencies, which reduces the risk exposure.

Once again we stress the importance of remembering this is **not** an all-or-nothing exercise. Our experience is the incremental approach works very well when proper evaluation of critical path and inter-dependencies are taken into account.

z/OS: Call COBOL from Java Batch

On z/OS there is a tremendous amount of business value invested in existing z/OS COBOL batch programs. Many wish *not* to replace that COBOL, but rather to *use it effectively within a Java batch process*. The question that remains is then how best to call the existing COBOL modules.

WebSphere Java Batch on z/OS has a feature called the "COBOL Container." It is designed to let you call and run COBOL modules *in the same z/OS address space* as your Java batch programs[6]. That makes calling the COBOL very efficient, with very low latency.

In addition, the COBOL container allows JDBC Type 2 connections to DB2 to be *shared between the Java batch container and the COBOL*, with *transactional context maintained* between the two. That allows the Java batch component to cooperate with the COBOL in a logically consistent transactional manner.

The value statement here is all about effective and efficient re-use of existing COBOL assets within a Java batch framework. The COBOL Container feature of WebSphere Java Batch on z/OS is able to provide this because of the design of z/OS itself. This is another example of how WebSphere Application Server for z/OS takes advantage of the platform[7].

For more on the COBOL Container, see the WP101783 Techdoc at `ibm.com/support/techdocs` and look for the COBOL Container PDF under "Specific Feature Brochures".

Summary

The message of this short document is that adopting WebSphere Java Batch as the execution framework for Java Batch does *not* require wholesale disruptive change to your environment. There are in fact several *incremental* approach patterns. The objective in the incremental approach is reduced complexity, better chances for project success, and limiting the risk of disruption by limiting the scope of each incremental change.

And remember: existing enterprise scheduler programs may integrate with WebSphere Java Batch on any supported platform using WSGRID. The ability to integrate Java and non-Java at the enterprise scheduler provides a way to create higher-level batch process orchestration flows.

WebSphere Java Batch provides value to your business in many different ways. With proper planning, concerns about disruptive adoption should not be part of your thinking. Proven incremental approach patterns exist. Take advantage of them to take advantage of the power of WebSphere Java Batch.

More Details

See the WP101783 Techdoc at `ibm.com/support/techdocs`. That Techdoc website has many presentations, papers and documents related to WebSphere Java Batch.

For more, contact Jeff Summers, Product Manager - WebSphere Application Foundation, at `summerje@us.ibm.com`

End of Brochure

5 A term used frequently in project management and applicable to batch processing as well. The *critical path* is the longest necessary sequence of activities that must be completed to complete the overall project.

6 Or, for those familiar with WAS z/OS, in the servant region where the Java batch program executes.

7 WebSphere Java Batch is built on top the WebSphere Application Server foundation. WAS z/OS has many points of platform integration where value derives to the application above by the way the platform features are exploited. For more on this, see WP101532 at `ibm.com/support/techdocs`.