"Modular Java - Generating Adaptable Software Having OSGI and Planting Season"

Rajeev

Research Scholar, Shri Venkateshwara University, UP

Abstract – With the rapid changes that happen in the zone of Web innovations, the porting and accommodation of existing Web applications into new stages that exploit up to date advances has turned into an issue of expanding criticalness. This paper displays a reengineering framework whose target system is a construction modeling based on the Model-View-Controller (MVC) configuration design and empowered for the Java Platform, Enterprise Edition (J2ee). The proposed framework is principally concerned with the decay of a legacy Web application by recognizing software parts to be changed into Java protests, for example, Java beans, Java server Pages (JSP), also Java Servlet.

---♦---

INTRODUCTION

Delightful towards the galaxy regarding assessed Java!

Building and conveying solid applications is a relic of past times. Applications that are made out of some more diminutive, overall characterized modules are a greatly improved approach. By concealing outline and usage parts that are prone to change behind a stable API, every module is less demanding to look after, test, and get it. This at last influences the generally maintainability and testability of the entire application. Tragically, as of Java, Java's inherent offices for modularity are intensely constrained. Basic directions are modularized into techniques, which are then modularized into classes. Classes could be further gathered into bundles, which offer a powerless manifestation of modularization. However that is the place Java modularity closes. Java offers no methods for modularizing classes or bundles of classes into coarse-grained modules.

Where Java misses the point, Osgi steps in. Osgi is a framework specification that carries modularity to the Java stage. In this book we're set to see how Osgi can empower advancement of generally characterized, approximately coupled modules that might be gathered into complete applications. However before we escape, we should get a feel for the sort of issue that Osgi tackles by listening in on a discussion between two associates on their approach to lunch.

MODULARITY

Brian's issue is absence of modularity—or all the more exactly, his disappointment to distinguish his auto's modularity. Autos are not stone monuments they are made up of a few dissimilar and distinct parts. It's regularly more practical to swap out those parts when they need to be displaced or redesigned than to swap out the whole auto. Assuming that parts might be reinstated and updated in something as unbending as a car, then why not something so delicate as software?

I comprehend what you're considering. You're feeling that you as of recently plan your applications to be particular. You put your classes and interfaces into bundles composed by their capacity. You outline your application into utilitarian layers. You keep coupling low by abstracting that usefulness behind interfaces. Maybe you utilize a reliance infusion framework, for example, Spring to make it conceivable to swap out one execution class for an alternate. Furthermore you may have even broken your application into two or all the more exclusively assembled undertakings.

MODULARITY IN JAVA

Java archive (JAR) files are frequently considered the unit of modularity in Java. Sadly, in any case, JAR files give just a flimsy figment of modularity. A run of the mill JAR file is truly just a sending time comfort, giving a vessel for a given set of classes, interfaces, and different assets. As outlined in Figure, once a JAR is put into the class path, the JAR limits break up in addition to any thought of modularity. All of the JAR's substance sit in the application's class space alongside the substance of each other JAR file in the class path. Hence, every open class in the JAR file is approachable by every different class in the class space.

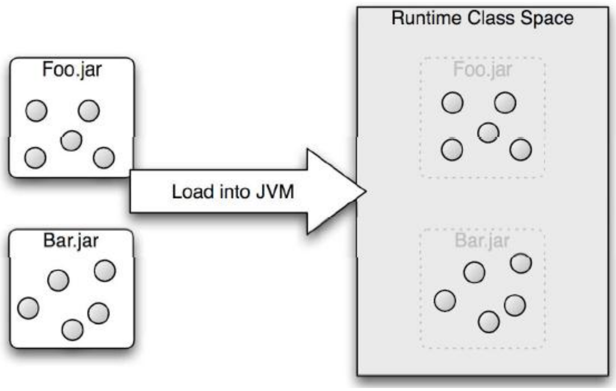

Figure: The limits gave by JAR files are counterfeit and blur away at runtime.

LAUNCHING OSGI

Osgi is a segment framework specification that carries modularity to the Java stage. Osgi empowers the making of remarkably firm, approximately coupled modules that might be made into bigger applications. What's all the more, every module might be independently created, tried, sent, redesigned, and dealt with insignificant or no effect to alternate modules. We should examine the elements that make up the OSGi specification also see how they help measured application advancement in Java.

At its least level, the Osgi specification characterizes an organization model for Java-based modules. The unit of arrangement in Osgi is known as a pack. Instead of make a totally new organization instrument, Osgi powers the existing JAR file position for groups. Osgi packs are much like normal JAR files, with the exception of that their META-Inf/manifest.mf file holds Osgi-specific metadata, including an absolute name, variant, conditions, and other sending parts.

When a pack is introduced into an Osgi framework, the Osgi life cycle oversees the status of the group. A group could be introduced, begun, halted, and uninstalled from the framework, emulating the life cycle recommended by the Osgi specification.

UNDERSTANDING THE OSGI PACKAGE

Right away that we've secured the profits of modularity and how Osgi carries modularity to Java, now is the right time to get down to the business of meeting expectations with Osgi. In this section, we're set to plunge our toes into the Osgi waters and see a portion of the fundamental stuff that goes into building an Osgi module. This will equip us for wading a little deeper into the waters of Osgi packages and administrations throughout the following few sections. Later, once we're adjusted to the basics of Osgi, we'll take a deep swoop into Spring-DM. In the first place things to begin with, in any case. We should begin by tinkering with a few the most famous Osgi frameworks, Equinox and Felix, to see what makes them tick.

All Osgi-based applications run inside an Osgi compartment (at times known as an Osgi framework). There are a few open source and business Osgi compartments to browse, including the accompanying:

- Eclipse Equinox
- Apache Felix (once Object web Oscar)
- Knopflerfish
- Concierge

Each of these holders has its advantages and disadvantages, however generally you're free to pick the holder that you like best and that comes with a permit that fits your needs. Equinox and Felix are presumably the two most prevalent Osgi holders accessible, so how about we begin by taking each of them for a test drive.

A WAB can't be a fragment group, however it can go about as a host for other fragment groups. A fragment group does not have its own particular class loader. All the fragments joined to a host impart the class loader of the host group. The Osgi specification has overall characterized decides that represent how a fragment Bundle-Class path is consolidated with host Bundle-Class path to come up with the powerful Bundle-Class path. While mapping the substance of a WAB to Web Application, powerful Bundle-Class path is utilized, which means Bundle-Class path of all the appended fragments are immediately considered. Despite the fact that fragments are not an extraordinary sample of modularity, they can be utilized to give extra substance as examined under.

LOOKING FOR JAR RECORDS

How often have you spent some minutes creating the following incredible bit of software just to be gone up against with the cryptic Class- Not Found Exception after submitting your work to the compiler? You're a gifted Java programmer. You know what to do, correct? The result needed here is basically a matter of adding some JAR file to your class path. Be that as it may which one? There are such a variety of Java classes scattered crosswise over such a large number of Java libraries, by what method would you be able to know beyond any doubt which JAR file you may as well include? What's more regardless of the possibility that you think you know which JAR file is needed, where do you head off to get it? To begin, how about we contemplate the characteristics that the application needs.

At an exceptionally fundamental level, we're set to need the accompanying:

- Something that creeps around one or more Maven archives, discovering JAR files to add to the record
- A method of indexing meta-information around a JAR file so it might be discovered later
- Some realm object(s) that speak to the JAR file by holding the meta-information
- An approach to question for JAR files that match certain criteria
- A web front close with the goal that the Java-creating masses can utilize the application to discover their libraries

A percentage of the information that may be intriguing around a JAR is the emulating:

- The URL of the Maven archive where the JAR was found.
- The Maven bunch ID, curio ID, and adaptation of the JAR.
- A rundown of the JAR's substance (e.g., a rundown of the .class files held inside the JAR).
- It could be decent to know if the JAR is a legitimate Osgi group (that is, does its META-Inf/manifest.mf have a Bundle-Symbolic Name header?).

In the event that this were an ordinary application, we'd most likely give each of the segments its own particular bundle inside the generally application structure and call that modularity. In the event that we're in an especially illuminated mood, we may even bundle every segment into its own particular JAR file that is eventually encased in the web application's WAR file.

Anyway this isn't a normal application. So as to profit from simplicity of organization, forming, parallel advancement, testability, and the other temperance's of modularity, we're set to manufacture this application as a gathering of Osgi groups.

As opposed to putting Maven Spider in the awkward position of needing to deal with the comings and goings of the file administration, we will utilize an administration tracker. Administration trackers hold the sum of the enchantment to keep track of if an administration is accessible, and they stow away the unpredictability of managing the Osgi administration registry through easier level Apis. Expert Spider is given an administration tracker that keeps track of the list administration also, upon appeal through the get service ( ) technique, gives the file benefit so we can add a Jar file to the list.

Despite the fact that the administration tracker modified works away any repulsiveness of managing the administration registry's low-level Apis, get service ( ) could still return invalid if the administration is occupied. Thus, we will need to check for an invalid administration before calling add jar file ( ). Anyhow in the event that you'd rather hold up for the administration to get accessible, we could call wait for service ( ) rather than get service ( ).

USING THE SERVICES OF BUNDLES

Osgi is about modularity. Also the unit of modularity in Osgi is a group. As we've as of recently examined, Osgi groups are little more than great dated JAR files with a tad bit of additional information in their manifests.

We've recently made a couple of straightforward groups. Anyhow now we're primed to turn it up a score and create a couple of additional reasonable packages that will meet up to structure the Dude, Where's My JAR? Application In this section we're set to make the first package for the application. It will be a straightforward package that exports just a solitary bundle and does not distribute or devour any administrations. Despite the fact that its basic, we'll confront a few fascinating issues, incorporating how to manage conditions on outsider JAR files that aren't Osgi-prepared.

Glassfish Server empowers communication between Osgi administrations and Java EE segments. This association is bi-directional. Osgi administrations oversaw by Osgi framework can conjure Java EE segments oversaw by Java EE compartment and the other way around. Application designers can definitively export Ejbs as Osgi administrations without needing to compose any Osgi code. That permits any immaculate Osgi part, which is running without the Java EE connection, to uncover the EJB and conjure it. That permits designers to compose business parts as Ejbs so they can take preference of Java EE stage characteristics, for example, decisive security, transaction administration, reliance infusion, and so on., but then permit them to be receptive to non-Java EE segments. Thus, Java EE parts can find Osgi administrations gave by non-Java EE Osgi groups also utilize them too. Glassfish Server augments the stage default infusion framework called Setting and Dependency Injection (SDI) framework to make it a great deal less difficult for Java EE segments to expend dynamic Osgi benefits in a type-safe manner.

We utilized Pax Construct's pax-make venture script to make a top-level undertaking for our application. Notwithstanding we'll utilization the pax-make pack script to produce a group subproject to convey the dominion questions in the application. As it stands, Jar file is a sufficient class for holding JAR file metadata. Yet in any case we'll need to compose the information kept in Jar file to an list that might be looked upon. In the following part, we'll construct the list administration, which utilizes an open source seek framework known as Compass to do the indexing and seeking. Meanwhile, we can go ahead and comment the classes with information that Compass can utilization the point when indexing a Jar file. In spite of the fact that we don't create software ideally, trust is not lost if your Osgi-built application depends with respect to a library that doesn't give an Osgi pack. One approach to alter the Compass group is to expand it, alter its manifest, and after that reconstitute the substance go into another JAR file. In any case that seems a touch great, especially assuming that its a manual exertion.

Rather, how about we consider two less amazing approaches to carry non-bundle jars into our Osgi application:

• Embed the JAR files inside the packages that need them.

• Wrap the JAR files with an Osgi manifest.

INTERNET FRAGMENT AND WAB

With presentation of segment characterizing annotations in Servlet 3.0 specification, the need for the web.xml has been diminished breathtakingly, yet some of the time it is fundamental. Prior, web.xml used to be a solid file bundled in WEB-INF/ catalog of the Web Application. Servlet 3.0 presents something many refer to as web-fragment.xml. It could be thought as a coherent dividing of the web.xml. As the name prescribes, it is a fragment of web.xml. Every such fragment could be bundled inside a jar file in WEB-Inf/lib. There could be a lot of people such web fragments in a Web Application and there even exists an instrument to request them when clashing directions are available. As such, we have not talked about anything which makes it any more intriguing than for the vanilla WAR file. If there should be an occurrence of a WAB, web fragments need not be bundled inside the WAB itself. Web fragments can be some piece of fragment packs which can then append to the WAB utilizing the Fragment-Host trait. As should be obvious, this permits a much more stupendous degree of modularity than what is offered on account of a conventional WAR.

Web applications can additionally be introduced as conventional Wars through a manifest reworking methodology. Osgi Web Applications specification characterizes a convention plan called web bundle which might be used to introduce plain vanilla WAR files into the Osgi runtime. The corresponding URL handler changes the information WAR to a WAB by including vital manifest entrances. After the change, the WAR carries on like a WAB. The change procedure might be altered by utilization of different question parameters in the URL. For a nitty gritty exchange on this, please allude to the Osgi Enterprise Specification.

As I've recently said, web groups are a peculiar breed. In numerous ways, they look like an accepted WAR file, however they likewise hold an Osgi-prepared manifest to empower them to be sent in an Osgi framework. Yet in spite of the fact that adding a manifest to a WAR file may make it an Osgi package, that alone doesn't completely exploit the profits of Osgi. A commonplace WAR file holds not just the web segment of an application yet additionally the complete purpose of the application. Regardless of the fact that the application is created in a measured manner, those modules wind up as JAR files.

Fragments are much like normal packages in that they are an unit of arrangement in Osgi. They're bundled as JAR files and are depicted in metadata in the META-Inf/manifest.mf file. Not at all like general groups, then again, are fragments futile independent from anyone else. They must be associated to an alternate pack.

Think about the relationship between packs and fragments as being similar to the relationship between a home stimulation system and Dvds. A regular advanced home diversion system likely incorporates in any event a television and a DVD player. Despite the fact that the amusement system is most likely helpful on its own for survey telecast programs, a DVD has little utility past that of a sparkling beverage napkin without the DVD player. By setting the DVD into the player, you give the DVD reason furthermore, in the meantime, augment the proficiency of the diversion system.

SPRING IN ADDITION TO OSGI

We made the file pack, which distributes an administration to the Osgi administration registry. Furthermore we devoured that administration from inside the spider package that slithers Maven vaults searching for JAR files. Little doubt remains that we have the vast majority of the pieces set up and that the main thing left to do is to fabricate the web front close to present the application to its clients.

However to begin with, we should consider what we needed to do to distribute and devour administrations. Distributed the list administration wasn't so awful we needed to make a pack activator, however in any event our work with the Osgi API was limited to the activator class. Then again, in the event that you're similar to me, you felt a spot grimy composing the Maven spider class that managed a Service tracker to find the list administration.

If there were somehow to compose our application code as Pojos and afterward pronounce that they are to devour or to be distributed as Osgi administrations.

In this section, we're set to see how to utilize Spring Dynamic Modules for Osgi (Spring-DM) to wipe out the sum of that Osgi-specific code that we used to distribute and devour administrations. In the meantime we'll carry all the force of the Spring Framework to Osgi. As opposed to customizing to the Osgi API, as we finished in the past part, this time we'll proclaim Spring beans to be Osgi administrations and infuse administrations into different beans.

Unless you've been existing under a rock or included in a multiyear lone restriction exercise, you've most likely caught wind of the Spring Framework. You may have even dealt with an undertaking or two that is dependent upon spring. It's a reality that spring has made an enormous effect on venture Java advancement.

Spring carries a considerable measure to the table for any application, including:

• Spring's backing for reliance infusion advertises detached coupling what's more high testability of application items.

• Spring's backing for aspect-turned modifying offers engineers a chance to divide cross-cutting concerns, for example, transactions, security, and reserving from center application code.

• Spring makes it simple to work with JDBC and other constancy frameworks, for example, Hibernate, JPA, and ibatis for information constancy.

• Spring backings decisive creation and utilization of remote administrations utilizing a mixture of removing choices, including RMI, Hessian, Burlap, and web administrations.

CONCLUSION

The quickened advancement of Web applications and the quick development of partnered Web advances have brought about a mixed bag of upkeep concerns. One of the significant support issues is the porting and acclimatization of existing Web applications into cutting edge Web-based innovations. In this paper, we have tended to this issue by applying reengineering strategies, including the source code examination, the software segment extraction, and the Web application re-calculating. We proposed a framework for incrementally moving legacy Web systems to new stages dependent upon J2ee advances. Utilizing this framework, a source application was refactored into JavaBeans configuration (Model), Java server Pages configuration (View), and Java Servlet group (Controller).

Future enlargements of the work introduced in this paper may keep tabs on the accompanying bearings: Firstly, we will examine the utilization of the wrapping innovation and the connector building design so as to coordinate all others language situations foreign made by the existing legacy Web system. Also, the use of a software bunching method will be inspected with the reason for recognizing strong aggregations of Web pages. A specific grouping calculation needs to be created keeping in mind the end goal to give the backing to the control combination of the vagrant applications. Thirdly, we will examine the method for programmed usage of examining element Web parts that must be structured at run time, for example, HTML structure filling, and database questioning. At last, we will deal with the era and utilization of the Undertaking Java Beans (EJB) innovation, especially element beans, for the back-finish execution of SQL statements.

REFERENCES

• Amanda W. Wu, Haibo Wang, and Dawn Wilkins, “Performance Comparison of Alternative Solutions for Web-To-Database Applications”, in Proceedings the Southern Conference on Computing, the University of Southern Mississippi, October 2000.

• Apache Felix OSGi implementation: http://felix.apache.org

• Carmine Albanese, Thierry Bodhuin, Enrico Guardabascio and Maria Tortorella, “A Toolkit for Applying a Migration Strategy: a Case Study”, in Proceedings of the 6th European Conference on Software Maintenance and Reengineering, 2002.

- Christy Lu, “A C to RPG Program Transformation Tool”, M.Sc Project, University of Waterloo, Department of Electrical & Computer Engineering, 1998.

- Cornelia Boldyreff and Richard Kewish, “Reverse Engineering to Achieve Maintainable WWW sites”, IEEE 2001.

- OSGi Module System: http://www.osgi.org

- OSGi Specifications: http://www.osgi.org/Specifications/HomePage

- Terence C. Lau, Jianguo Lu, John Mylopoulos, Kostas Kontogiannis, “The Migration of Multi-tier E-commerce Applications to an Enterprise Java Environment”, Information System Frontiers, 5:2, pp. 149-160, 2003.

- Thierry Bodhuin, Enrico Guardabascio and Maria Tortorella, “Migrating COBOL Systems to the WEB by using the MVC design pattern”, in Proceedings of the 9th Working Conference on Reverse Engineering, 2002

- Vesselin Ivanov, “Moving to a Java object environment: Best practices of WebSphere Commerce migration and LOQS”, December 2002. http://cas.ibm.com/toronto /publications/TR-74.188 /27/ivanov.pdf

- Ying Zou, Kostas Kontogiannis, “A Framework for Migrating Procedural Code to Object-Oriented Platforms”, in Proceedings of the 8th IEEE Asia-Pacific Software Engineering Conference, pp. 408-418, Macau, China, December 2001.