JBoss Microcontainer Reference

A Reference for Developers

JBoss Microcontainer Reference: A Reference for Developers

2.0.0

Table of Contents

Target Audience

This reference is aimed at microcontainer developers.

It aims at describing how the microcontainer is constructed, the design decisions made and how to extend or develop it.

Preface

Commercial development support, production support and training for the JBoss Microcontainer is available through JBoss Inc. [http://www.jboss.com] The JBoss Microcontainer is a project of the JEMS product suite.

Authors:

- Adrian Brock - JBoss Microcontainer Project Lead and Chief Scientist of JBoss Inc.

  Scott Stark - VP Architecture & Technology

  Kabir Khan - JBoss AOP Project Lead

  Ales Justin - JBoss Core Developer

Chapter 1. Outline of Reference Guide

An outline of the topics/order we should cover.

- Introduction of the JBoss Microcontainer and its purpose.
- Overview of the JBoss Microcontainer modules and what the various dependencies between them are.
- Setup/bootstrap of a kernel. Discussion of various bootstrap coding/configuration to setup a kernel for different environments/purposes. This should be somewhat of a deeper dive overview in that it will reference other modules and SPIs that are used for the setup.
  - Unit testing
  - Embedded J2SE
  - Embedded (EJB3, JBPM, SEAM, ...). How other projects can leverage the JBoss Microcontainer for configuration, IoC, ...
  - JBossAS
  - JBossESB
- Overview of the JBoss Microcontainer core apis and their relations;
  - container: Configuration, ClassAdapter, ControllerContex, BeanInfo, Joinpoint, MetaDataRepository, MetaData, ...
  - dependency: Controller, ControllerContex, DependencyInfo, DependencyItem, ...
  - kernel: Kernel, KernelFactory, KernelConfigurator, KernelController, KernelControllerContext, KernelDeployment, KernelInitializer, KernelEventManager, KernelMetaDataRepository, KernelBus KernelRegistry ...
  - aop-mc-int: AOPJoinpointFactory, AspectBinding, Aspect, MixinBinding, StackBinding, lifecycle stuff, ...
  - deployers: MainDeployer, Deployer, DeploymentUnit, Attachments, DeploymentContext, StructureMetaData, StructuredDeployers, StructureBuilder, ManagedObjectBuilder, ...
  - managed/metatype: ManagedObject, ManagedProperty, Fields, MetaType, SimpleMetaType, CompositeMetaType, MetaValue, ...

Chapter 2. Wiki Links

A collection of useful wiki/forum links to help pull together content.

- MC High Level Road Map [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMicrocontainerHighLevelRoadMap]
- Executive Overview [http://www.jboss.com/index.html?module=bb&op=viewtopic&t=59868]
- JBossMicrocontainer [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMicrocontainer]
- JBossController [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossController]
- JBoss5 High Level Road Map [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBoss5HighLevelRoadMap]
- JBossAS5 Virtual Deployment Framework [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBoss5DeploymentFramework]
- FAQ [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMicrocontainerFAQ]

Chapter 3. Introduction to the JBoss Microcontainer

The JBoss Microcontainer (MC) provides an environment to configure and manage POJOs (plain old java objects). It is designed to reproduce the existing JBoss JMX Microkernel but targetted at POJO environments. As such it can be used standalone outside the JBoss Application Server.

The 2.0.x release provides a set of features to support POJO deployment in JBossAS-5.0.x, JBossESB, and the bootstrap of services used by standalone projects like embedded EJB3.

This document takes you through the design of the microcontainer including design decisions and future directions.

JBoss Microcontainer Highlights

The primary focus of the MC is as a framework for building server kernels. It replaces the legacy JMX based kernel found in JBossAS4.x and earlier with a POJO based kernel that has been generalized to better support extensibility along the primary aspects required for a server type of environment: aop, metadata, class loading, deployments, state management, lifecycle/dependcies, configuration, and management. The use of the term server should not be correlated with large memory/cpu requirment environments of typical application/web servers. A server in the context of the MC is just a kernel for plugging POJO container models into. The MC can be configured for extremely lightweight application type setups like a junit testcase as well as full feature application servers.

Information and Roadmaps

A high level roadmap for the MC and status can be found in the wiki page JBossMicrocontainerHighLevelRoadMap [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBossMicrocontainerHighLevelRoadMap]. A more detailed roadmap of the issues going into releases can be found in the MC JIRA [http://jira.jboss.com/jira/browse/JBMICROCONT?report=com.atlassian.jira.plugin.system.project:roadmap-panel] project page.

Design discussion occur in the JBoss POJO Developers Forums [http://www.jboss.com/index.html?module=bb&op=main&c=18]. This currently includes 3 sub-forums: Design the new POJO MicroContainer [http://www.jboss.com/index.html?module=bb&op=viewforum&f=204] (the core MC projects independent of any particular usage), Design of POJO Server [http://www.jboss.com/index.html?module=bb&op=viewforum&f=208] (discussions related to application server environments like JBossAS, JBossESB), Design of OSGi Integration [http://www.jboss.com/index.html?module=bb&op=viewforum&f=256] (discussion of how a kernel/framework like OSGi should integrate with the MC).

An overview of the MC modules

This section introduces the various MC modules

aop-mc-int

Integration between the JBossAOP and MC projects

container

A better name would be joinpoint. This module contains: reflection, the integration point for manipulating class information at runtime, e.g. overriding annotations or obtaining an aop instance advisor. joinpoint, the joinpoint model including the join point factory. classadaptor, the integration and configuration spi. metadata, base metadata types and repository

dependency

Dependency management is handled by the controller. The controller is the core component for keeping track of contexts to make sure the configuration and lifecycle are done in the correct order including dependencies and classloading considerations.

deployers

Deployers load components from various models, POJOs, JMX, Spring, Java EE, etc. into the MC runtime.

kernel

The kernel defines the core kernel spi including, boostrap, configuration, POJO deployments, dependency, events, bean metadata, and bean registry.

managed

The managed module defines the base objects defining the management view of a component.

metatype

The metatype module defines the base types found in the management view of a component.

osgi-int

This contains the integration classes that adapt the OSGi model onto the MC.

spring-int

This contains the integration classes that adapt the spring model onto the MC.

Chapter 4. Reflection Model

WARNING: The reflection model is subject to change without notice. It is currently under developement as issues with AOP integration and using the Microcontainer in more constrained environments are investigated.

The reflection model provides an abstraction over java.lang.reflect. It exists in the container module in package org.jboss.reflect.

Aims

Cache

Reflection can be a bottleneck unless the information is cached. This package provides a single stop cache for reflection data rather than repeating this behaviour throughout JBoss.

Abstraction

It is anticapted that the reflection model can be improved in different circumstances or environments. This module aims to be the one place where those issues are resolved.

Hide implementation

Under AOP, classes can change their characteristics usually through some sort of bytecode manipulation. This module aims to "hide" some of these details from routines that "don't care" about these implementation details.

Issues

Java5

Complete the support for annotations and generics.

Lazy Cache

Improve the lazy cache implementation. Details should only be retrieved when accessed.

AOP

It is still an open issue whether this implementation can be used by AOP or whether the ClassAdapter is a better place.

Javassist

The current implementation just uses simple reflection. Javassist can proably provide a better implementation including supporting annotations before java5?

Chapter 5. Joinpoint - Accessing a Bean

WARNING: The joinpoint model is subject to change without notice. It is currently under developement as issues with AOP integration and using the Microcontainer in more constrained environments are investigated.

The joinpoint model provides a mechanism to access class operations in a standard way.

Aims

Stanardize reflection operations

There are many places that use reflection within JBoss. The joinpoint provides a mechansim to deal with reflection dispatching without having worry about all the standard error handling, etc.

Overridding Definition

The joinpoint allows the mechanism of dispatch to be overridden either through configuration or implementation. Examples would be not using get/set to define a property either because it is a different method name or it should use direct field access.

Issues

javabean versus object

Much of the javabean abstraction has moved the BeanInfo model, it is questionable whether the ability to override on a plain object is still required.

Alternative implementations

One use of this abstraction would be to precompile reflection objects using javassist for use in environments where dynamic reflection is not supported.

AOP

Another potential other use is to "optimize away" reflection when already dispatching requests through AOP.

Chapter 6. Dependency Controller

WARNING: The dependency controller is subject to change without notice. It is currently under developement as issues with the integration of the JMX microkernel and microcontainer are resolved.

Chapter 7. Dependency Info - describing dependencies

WARNING: The dependency controller is subject to change without notice. It is currently under developement as issues with the integration of the JMX microkernel and microcontainer are resolved.

Chapter 8. Bootstrapping the Kernel

The bootstrap provides a simple mechanism for bootstrapping the microcontainer.

Bootstrap - bootstrapping

This bootstrap creates and configures the microcontainer. This involves creating a KernelConfig from which the KernelFactory can be instantiated using the KernelInstantiator.

Config - bootstrap configuration

This KernelConfig provides the mechanism to configure the bootstrap including how the core kernel services are instantiated and configured.

KernelFactory - kernel instantiation

The KernelFactory is responsible for initializing and configuring the kernel services using information provided by the KernelConfig.

Chapter 9. Registering Objects

The registry is used to register objects against logical names.

Beans can either be registered directly with the registry or KernelRegistryPlugins can be registered to supply objects from other registries, e.g. JNDI.

KernelRegistry - retrieving named objects

The kernel registry is used to register and retrieve objects against a logical name.

When the object registered is an instance of KernelRegistryPlugin it will be asked for an entry if it is not directly registered with the registry. The registry factories are asked in turn in the order they were registered.

KernelRegistryPlugin - helpers for the registry

A registry factory allows other registry domains (such as JMX or JNDI) to be inserted into the kernel registry.

To insert a registry factory into the registry, simply register it like any other object.

Chapter 10. The Invocation Bus

The invocation bus provides detyped, named access to the objects in the registry.

Bus - invoking objects by name

The invocation bus provides

- Detyped invocations - the beans operations can be invoked by name using
- Named endpoints - the target bean is specified by a name registered in the registry

Chapter 11. Describing a Bean

The BeanInfo provides a description of a POJO. This information allows the microcontainer to know what can be done with the bean.

This is similar to the BeanInfo provided by the java.beans package but some java environments like J2ME do not include this package because it has dependencies on AWT.

BeanInfoFactory - retrieveing bean descriptions

The `BeanInfoFactory` is used to create BeanInfos for a given class.

```
BeanInfo getBeanInfo(String className)
```

The default implementation is the IntrospectionBeanInfoFactory which uses reflection.

BeanInfo - information about a bean

The BeanInfo provides top level information and access to the more detailed information about the bean.

PropertyInfo - information about a property

The PropertyInfo describes an property, i.e. something that follows the

```
Type getX(); void setX(Type x)
```

javabean pattern.

EventInfo - information about an event

The EventInfo describes an event.

Chapter 12. Decribing Beans

The BeanMetaData provides configuration used to deploy beans.

This is similar to the -service.xml files in JBoss4.

BeanMetaData - deploying a single bean

The BeanMetaData provides top level information and access to the more detailed information about the bean deployment.

ClassLoaderMetaData - configuring a classloader

The classloader metadata is used to configure the classloader for an individual bean or a whole deployment.

PropertyMetaData - configuring properties

The property metadata is used to configure individual bean property with values.

ParameterMetaData - configuring values passed to constructors or methods

The parameter metadata is used to pass values to constructors and methods.

SupplyMetaData - specifying the bean supplies a demand

The supply metadata is used to define what the bean supplies.

DemandMetaData - specifying the bean demands a dependency

The demand metadata is used to define what the bean demands.

ValueMetaData - configuring values

The value metadata is used to define values.

DependencyValueMetaData - defining named dependencies

An extension to the value metadata that treats the underlying value as the name of an entry in the Registry and returns the registry's registered object when asked for the value.

Chapter 13. Bean Configuration - instantiating and configuring beans

The kernel configurator provides a service to instantiate and configure beans.

KernelConfigurator - configuring beans

This service is used by the Controller once it knows the dependencies have been satisfied. ControllerContext

Chapter 14. Bean Dependency - wiring beans together

The POJO controller extends the dependency controller to provide a specific lifecycle for POJOs.

KernelController - wiring beans

The POJO controller defines the following state operations for

describe: determines the dependencies from the bean metadata

instantiate: invokes the constructor or factory

configure: configures the bean

create: invokes any "create()" method

start: invokes any "start()" method

install: registers the bean with the registry

KernelController - demand and supply

The POJO controller acts as a Registry plugin to handle demand and supply.

Chapter 15. Bean Deployment - deploying beans

<authorblurb>

Scott Stark
</authorblurb>

The bean deployer provides routines for deploying beans. TODO: translate/ expand the wiki into this doc. JBoss5DeploymentFramework [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBoss5DeploymentFramework]

Design Overview

The JBoss5 virtual deployemnt framework takes the aspect oriented design of many of the earlier JBoss containers and applies it to the deployment layer. It is also based on the POJO microntainer rather than JMX as in previous releases. For an overview of the classes as a UML diagram see JBoss5DeploymentFrameworkClasses. The key classes in that diagram are:

MainDeployer — this interface defines the contract for the MainDeployer. The MainDeployer handles parsing of deployment archives into Deployment instances and deployment of those instances into the microcontainer. This in is an update of the JMX based MainDeployer from earlier versions to a one based on the Microcontainer, JBoss5VirtualFileSystem, and Virtual Deployment Framework(VDF). Deployers are registered with the MainDeployer as an ordered list of deployers. MainDeployer contains two sets of deployers:

- StructureDeployers : used to analyze the structure of a DeploymentContext when addDeploymentContext(DeploymentContext) is invoked. For each StructureDeployer the determineStructure(DeploymentContext) method is invoked to analyze the deployment. A StructureDeployer? returns true to indicate that the deployment was recognized and no further StructureDeployer should analyze the DeploymentContext.
- Deployers used to translate a DeploymentUnit into a runtime kernel beans when the MainDeployer.process is run. The Deployer methods are:
  - boolean isRelevant() :does the deployer want to process the unit
  - prepareDeploy() : take the new deployment to the ready stage
  - prepareUndeploy() : get ready to undeploy
  - handoff(new, old) : handover control from new to old
  - commitDeploy() : new deployment is now in control
  - commitUndeploy() : old deployment is out of here
  - getRelativeOrder() : specify the relative order of the deployer in a chain

DeploymentUnit — a representation of a runtime unit of work a Deployer operates on.

DeploymentContext — a representation of structural aspects of deployable content.

ManagedObject	a representation of the manageable properties for a deployment.
VFS	the api for representing the read-only file system of the deployment.
VirtualFile	the api for a file in the deployment.
DomainClassLoader	A generalization of the legacy JMX based unified class loading model. This is still in progress. The org.jboss.vfs.classloding.

Current StructureDeployers JBoss5StructureDeployerClasses [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBoss5StructureDeployerClasses]

- org.jboss.deployers.plugins.structure.vfs.AbstractStructureDeployer
- org.jboss.deployers.plugins.structure.vfs.explicit.DeclaredStructure
- org.jboss.deployers.plugins.structure.vfs.file.FileStructure
- org.jboss.deployers.plugins.structure.vfs.jar.JARStructure
- org.jboss.deployers.plugins.structure.vfs.war.WARStructure
- org.jboss.deployment.EARStructure

Current Deployers

- org.jboss.deployers.plugins.deployers.kernel.BeanDeployer
- org.jboss.deployers.plugins.deployers.kernel.KernelDeploymentDeployer
- org.jboss.deployers.plugins.deployers.kernel.BeanMetaDataDeployer
- org.jboss.deployers.plugins.deployers.kernel.BeanMetaDataDeployer
- ServiceDeployments [http://wiki.jboss.org/wiki/Wiki.jsp?page=ServiceDeployments]
  - org.jboss.system.deployers.SARDeployer
  - org.jboss.system.deployers.ServiceClassLoaderDeployer
  - org.jboss.system.deployers.ServiceDeploymentDeployer
  - org.jboss.system.deployers.ServiceDeployer
- JBoss5WebDeployments [http://wiki.jboss.org/wiki/Wiki.jsp?page=JBoss5WebDeployments]
  - org.jboss.deployment.WebAppParsingDeployer
  - org.jboss.deployment.JBossWebAppParsingDeployer
  - org.jboss.web.tomcat.tc6.deployers.TomcatDeployer

- org.jboss.resource.deployers.RARDeployer
- org.jboss.resource.deployers.RARParserDeploye

Deployer Helper/Base Classes JBoss5BaseDeployerClasses

AbstractDeployer

Chapter 16. Events - notifications from the microcontainer and services

The event subsystem allows services to produce and consume events.

Event - event abstraction

The Event class is intended to abstract the different event mechanisms used by different software.

EventEmitter - producting events

Event emitters produce events and send them to their registered listeners.

Listeners can be registered with a filter to filter the events they are interested in before they are sent.

EventListener - consuming events

Event listeners consume events from emitters.

EventFilter - filtering events

A filter allows events to filtered before they are sent. This is can be more efficient than filtering the event in the listener.

EventManager - named listening

The event manager allows emitters and listeners to be more detached from each other. Instead of direct registration, the names of the objects in the registry are used.