Snowdrop Guide

Joshua Wilson

Version 3.1.2
2014-12-16

Table of Contents

This book is a guide for using Snowdrop with JBoss AS.

1. Introduction to Snowdrop

Snowdrop is a utility package that contains JBoss-specific extensions to the Spring Framework. These are:

- extensions to Spring Framework classes that can be used wherever the generic implementations provided by the framework do not integrate correctly with JBoss Application Server.
- extensions for deploying and running Spring applications with the JBoss Application Server.

The Snowdrop distribution contains two types of artifacts:

Utility archives

Libraries that contain utility classes and can be packaged in applications

Server-side components

Pre-packaged components that can be installed in the application server as subsystems.

The server-side components may include some of the utility libraries, where they are needed by the deployer or, respectively, subsystem.

Snowdrop provides the following utility libraries:

`snowdrop-vfs.jar`

A library that contains the support classes for resource scanning (scanning the classpath for bean definitions, or using "classpath*:" -style patterns).

`snowdrop-weaving.jar`

A library that contains the support classes for load-time weaving.

`snowdrop-interceptors.jar`

A library that contains utility classes used internally by the Spring deployer. It is not intended for direct use by developers.

`snowdrop-namespace.jar`

Custom namespace for easier configuration of beans that need access to JBoss internals (JMX server locator, JCA activation spec and resource adapter for JMS endpoints)

Snowdrop provides the following server-side pre-packaged deployers and subsystems:

`snowdrop-subsystem-as7.zip`

This is a packaged distribution of the Spring subsystem for JBoss Application Server 7, which bootstraps and registers the application contexts to be used by your Java EE applications.

2. Snowdrop Usage

This chapter details how to use each of the components included in

2.1. VFS-enabled Application Contexts

NOTE	From Spring 3.0 onward, the `ApplicationContext` implementations shipped with the Spring framework are VFS-compatible. The components described in this section are included with Snowdrop to provide backwards compatibility, but are not necessarily required.

The `snowdrop-vfs.jar` library supports resource scanning in the JBoss Virtual File System (VFS). It must be included in Spring 2.5 based applications that use classpath and resource scanning.

When the Spring framework performs resource scanning, it assumes that resources are either from a directory or a packaged JAR and treats any URLs it encounters accordingly.

This assumption is not correct for the JBoss VFS, so Snowdrop provides a different underlying resource resolution mechanism by amending the functionality of the `PathMatchingResourcePatternResolver`.

This is done by using one of two `ApplicationContext` implementations provided by `snowdrop-vfs.jar`:

`org.jboss.spring.vfs.context.VFSClassPathXmlApplicationContext`

Replaces the Spring `org.springframework.context.support.ClassPathXmlApplicationContext`.

`org.jboss.spring.vfs.context.VFSXmlWebApplicationContext`

Replaces the Spring `org.springframework.web.context.support.XmlWebApplicationContext`.

In many cases, the `VFSClassPathXmlApplicationContext` is instantiated on its own, using:

```
ApplicationContext context =
new VFSClassPathXmlApplicationContext("classpath:/context-definition-file.xml");
```

The `XmlWebApplicationContext` is not instantiated directly. Instead, it is bootstrapped by either the `ContextLoaderListener` or the `DispatcherServlet`. Both classes have configuration options that allow users to replace the default application context type with a custom application context type.

To change the type of application context created by the `ContextLoaderListener`, add the `contextClass` parameter as shown in the following example code:

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:spring-contexts/*.xml</param-value>
</context-param>
<context-param>
    <param-name>contextClass</param-name>
    <param-value>
        org.jboss.spring.vfs.context.VFSXmlWebApplicationContext
    </param-value>
</context-param>
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

To change the type of application context created by the `DispatcherServlet`, use the same `contextClass` parameter on the `DispatcherServlet` definition as shown:

```
<servlet>
    <servlet-name>spring-mvc-servlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/mvc-config.xml</param-value>
    </init-param>
    <init-param>
        <param-name>contextClass</param-name>
        <param-value>
            org.jboss.spring.vfs.context.VFSXmlWebApplicationContext
        </param-value>
    </init-param>
</servlet>
```

IMPORTANT

`ZipException`

If you encounter the `ZipException` when attempting to start the application, you need to replace the default `ApplicationContext` with one of the VFS-enabled implementations.

```
Caused by: java.util.zip.ZipException: error in opening zip file
...
at
org.springframework.core.io.support.PathMatchingResourcePatternResolv
er
.doFindPathMatchingJarResources(PathMatchingResourcePatternResolver.j
ava:448)
```

2.2. The JBoss Custom Namespace

Snowdrop includes the `snowdrop-namespace.jar` which adds the custom namespace, `jboss`, to support Spring on JBoss Application Server. The goal of this custom namespace is to simplify the development of Spring applications that run on JBoss, by reducing the amount of proprietary code and improving portability.

The amount of proprietary code is reduced by replacing bean definitions that include references to specific JBoss classes with namespace-based constructs. All the knowledge about the proprietary classes is encapsulated in the namespace handlers.

The applications are more portable because certain proprietary classes may change when upgrading to a different version of the application server. In such cases, the runtime will be detected automatically by Snowdrop which will set up beans using the classes that are appropriate for that specific runtime.

Set up the custom namespace as follows:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:jboss="http://www.jboss.org/schema/snowdrop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.jboss.org/schema/snowdrop
                        http://www.jboss.org/schema/snowdrop/snowdrop.xsd">
</beans>
```

2.2.1. Accessing the Default JBoss MBean Server

Access the default MBean server of JBoss AS as follows:

```
<jboss:mbean-server/>
```

The bean will be installed with the default id 'mbeanServer'. If necessary, developers can specify a different bean name:

```
<jboss:mbean-server id="customName"/>
```

IMPORTANT

The location of the MBean server has changed between versions of JBoss AS, so if using the following configuration fails with deployment errors:

```
<bean id="mBeanServer"
      class="org.jboss.jmx.util.MBeanServerLocator"
      factory-method="locateJBoss" />
```

The workaround for this issue is to use this configuration instead:

```
<bean id="mbeanServer"
      class="org.springframework.jmx.support.MBeanServerFactoryBean">
    <property name="locateExistingServerIfPossible" value="true" />
</bean>
```

2.2.2. JCA/JMS Support Beans

Spring JMS message listeners (including message-driven POJOs) can use a JCA-based MessageListenerContainer. The configuration of a JCA-based listener container in Spring requires the setup of a number of beans based on application-server specific classes. Using the JBoss custom namespace, set up the ResourceAdapter and ActivationSpec configuration as follows:

```
<jboss:activation-spec-factory id="activationSpecFactory"/>
<jboss:resource-adapter id="resourceAdapter"/>
```

This can be further used in a JCA message listener configuration:

```
<jms:jca-listener-container resource-adapter="resourceAdapter"
                            acknowledge="auto"
                            activation-spec-factory="activationSpecFactory">
    <jms:listener destination="/someDestination"
                  ref="messageDrivenPojo"
                  method="pojoHandlerMethod"/>
</jms:jca-listener-container>
```

2.3. Load-time weaving

NOTE	From Spring 3.0 onward, load-time weaving on JBoss Application Server is supported out of the box. The component described in this section can be used to facilitate backward compatibility, but configuring a custom load-time weaver is not required when using Spring 3.0 or later.

Load-time weaving support is provided by the `snowdrop-weaving.jar` library.

To perform load-time weaving for the application classes in Spring (either for using load-time support for AspectJ or for JPA support), the Spring framework needs to install its own transformers in the classloader. For JBoss AS, a classloader-specific `LoadTimeWeaver` is necessary.

Define the `JBossLoadTimeWeaver` in the Spring application context as shown here:

```
<context:load-time-weaver
    weaver-class="org.jboss.instrument.classloading.JBossLoadTimeWeaver"/>
```

2.4. The Spring Interceptor (Deployer)

The Spring Interceptor (previously called a "deployer") allows you to bootstrap a Spring application context, bind it in JNDI, and use it to provide Spring-configured business object instances.

2.4.1. JBoss + Spring + EJB 3.0 Integration

Snowdrop contains a JBoss interceptor that supports Spring packaging in JBoss AS. This means it is possible to create JAR archives with a `META-INF/jboss-spring.xml` file to have the Spring bean factories deploy automatically.

EJB 3.0 integration is also supported. Spring beans created in such archive deployments can be injected directly into an EJB by using the `@Spring` annotation.

2.4.2. Installation on JBoss Application Server 7

WARNING	**Deprecated** • Spring 2.5, 3.0.x and 3.1.x are deprecated. Support for deprecated versions of Spring will be dropped in Snowdrop 4.

2.4.2.1. Automatic Installation of Snowdrop

For easy installation of Snowdrop module, use the Snowdrop installer. The installer copies Snowdrop and Spring jars to their appropriate location within `${JBOSS_HOME}/modules` directory. The installer also creates a new `$JBOSS_HOME/standalone/configuration/standalone-snowdrop.xml` file based on `$JBOSS_HOME/standalone/configuration/standalone.xml` to register the snowdrop extension and

subsystem. You can run JBoss AS with this new configuration using `$JBOSS_HOME/bin/standalone.sh --server-config=standalone-snowdrop.xml`.

To install Snowdrop using the installer;

1. Download and Unzip `snowdrop-3.1.2.Final-install.zip`
2. On the command line, navigate to the `snowdrop-module-installer` directory
3. Execute the following command:

```
$ mvn package -DJBOSS_HOME=/path/to/jboss_home
```

By default, the installer installs Snowdrop 3.1.2 and Spring 4.1.4.RELEASE. To install a different version, execute the following command:

```
$ mvn package -P${desired-spring-version} -DJBOSS_HOME=/path/to/jboss_home
-Dversion.snowdrop=${desired-snowdrop-version}
```

Use one of the following spring version profiles: spring-2.5, spring-3, spring-3.1, spring-3.2, spring-4.0, and spring-4.1 *(the default)*.

2.4.2.2. Manual Installation of Snowdrop

Manual installation is usefull when you want to control the location of the deployment or if you want to embedded the JARs into your application.

To manually install the Snowdrop Deployment subsystem:

1. Download and Unzip `jboss-spring-subsystem-as7.zip` OR `jboss-spring-subsystem-as7-nodeps.zip`.
2. Copy the contents of the `module-deployer` directory to the `$JBOSS_HOME/modules/system/add-ons/snowdrop` directory of your JBoss AS installation.
3. Copy the contents of one of the `module-spring-2.5/`, `module-spring-3/`, `module-spring-3.1/`, `module-spring-3.2/`, `module-spring-4.0/` or `module-spring-4.1/` directories to the `$JBOSS_HOME/modules/system/add-ons/snowdrop` directory of your JBoss AS installation.
    a. If you use `jboss-spring-subsystem-as7.zip` then the `module-spring-*` will have the Spring JARs installed.
    b. If you use `jboss-spring-subsystem-as7-nodeps.zip` then the `module-spring- will *NOT` have the Spring JARs installed.

The above steps create two modules inside JBoss AS:

`org.jboss.snowdrop:main`

The module that contains the Snowdrop JARs.

`org.springframework.spring:snowdrop`

A module that contains the Spring JARs required by Snowdrop. depending on the previously chosen version it should contain JARs from only one of the following Spring versions: 2.5, 3, 3.1, 3.2, 4.0 or 4.1. Users may add other JARs to the module as needed. In which case, they need to adjust the `module.xml` file accordingly. It is a dependency of `org.jboss.snowdrop:main`

The Snowdrop Deployment subsystem does not contain Spring archives if you use `jboss-spring-subsystem-as7-nodeps.zip`. You will need to install them separately. Download the needed files from Maven Central, according to the files listed in the corresponding versions.

Spring 2.5.6.SEC03

- aspectjrt.jar
- aspectjweaver.jar
- aopalliance.jar
- spring-aop.jar
- spring-beans.jar
- spring-core.jar
- spring-context.jar
- spring-context-support.jar
- spring-web.jar

Spring 3.0.7.RELEASE and 3.1.4.RELEASE

- aspectjrt.jar
- aspectjweaver.jar
- aopalliance.jar
- spring-aop.jar
- spring-asm.jar
- spring-beans.jar
- spring-core.jar
- spring-expression.jar

- spring-context.jar
- spring-context-support.jar
- spring-web.jar

Spring 3.2.13.RELEASE, 4.0.9.RELEASE, and 4.1.4.RELEASE

- aspectjrt.jar
- aspectjweaver.jar
- aopalliance.jar
- spring-aop.jar
- spring-beans.jar
- spring-core.jar
- spring-expression.jar
- spring-context.jar
- spring-context-support.jar
- spring-web.jar

The final step in the installation is to change `$JBOSS_HOME/standalone/configuration/standalone.xml`. . Add `<extension module="org.jboss.snowdrop"/>` inside the `<extensions>` element. . Add `<subsystem xmlns="urn:jboss:domain:snowdrop:1.0"/>` inside the `<profile>` element.

2.4.3. Defining the JNDI name

You can specify the JNDI name explicitly by putting it in the description element of the Spring XML.

```
<beans>
    <description>BeanFactory=(MyApp)</description>
    ...
    <bean id="springBean" class="example.SpringBean"/>
</beans>
```

`MyApp` will be used as the JNDI name in this example.

2.4.4. Parent Bean factories

Sometimes the deployed Spring bean factory must be able to reference beans deployed in another Spring deployment. This can be done by declaring a parent bean factory in the description element in

the Spring XML, as follows:

```
<beans>
    <description>BeanFactory=(AnotherApp) ParentBeanFactory=(MyApp)</description>
    ...
</beans>
```

2.4.5. Injection into EJBs

Once an `ApplicationContext` has been successfully bootstrapped, the Spring beans defined in it can be used for injection into EJBs. To do this, the EJBs must be intercepted with the `SpringLifecycleInterceptor`, as in the following example:

```
@Stateless
@Interceptors(SpringLifecycleInterceptor.class)
public class InjectedEjbImpl implements InjectedEjb {
    @Spring(bean = "springBean", jndiName = "MyApp")
    private SpringBean springBean;

    /* rest of the class definition omitted */
}
```

In this example, the EJB `InjectedEjbImpl` will be injected with the bean named `springBean`, which is defined in the `ApplicationContext`.

A. Migration Notes

***This section of the guide will track any breaking changes introduced in new** and identify any steps required to accommodate those changes in your application.*

A.1. Snowdrop 3.1.2

A.1.1. Deployers and Interceptors

If you are using the Snowdrop API, specifically the snowdrop-deployers.jar, and you want to use the latest version of Snowdrop then refactor your project to use the snowdrop-interceptors.jar. Some older deprecated classes were removed but the classes used in the API have not changed, so this does not require any other modifications.

In case you do not have the possibility to rebuild your project you can replace or remove the appropriate packages directly from the archive. The archive then becomes compatible with the latest Snowdrop API.