JBoss WORLD
CHICAGO 2009

FOLLOW US:

TWITTER.COM/REDHATSUMMIT

TWEET ABOUT US:

ADD #SUMMIT AND/OR #JBOSSWORLD TO THE END OF YOUR EVENT-RELATED TWEET

presented by

JBoss WORLD CHICAGO 2009

Spring on JBoss

Marius Bogoevici

Senior Software Engineer, Red Hat

September 2nd, 2009

presented by

About the presenter:

- Marius Bogoevici - mariusb@redhat.com
- Senior Software Engineer at Red Hat
- Lead for Snowdrop, the JBoss/Spring utilities library
- Early Spring adopter - since 2004
  - Also commiter to Spring Integration (Spring-based EAI patterns implementation)

Outline

- What brings JBoss and Spring together?
- Project Snowdrop
- Using Spring on JBoss: a Java EE-based perspective
- Tooling
- Red Hat Open Choice Strategy and WFK

Spring is an integration framework

- Spring is, above all, an **integration framework**
- Easily integrate application components (POJOs) and runtime services
- Defines abstractions that interact with Java EE standard APIs
- Provides its own implementations for common middleware functionality (e.g. transaction management)
- Or, **uses application server services**

What brings Spring and JBoss together?

- Spring provides the **application development model**
  - dependency injection/configuration
  - templates
  - transaction demarcation
- JBoss provides the **runtime services**
  - web container, transaction coordinator, messaging middleware
  - connection pools, JMX management
  - classloading, deployment
- So, JBoss is a great place for running Spring applications!
- … and opportunities exist to provide a better experience with richer integration!

What is Snowdrop?

- A collection of utilities for Spring on JBoss:
  - the Spring Deployer (since 2006)
  - VFS integration
  - Load-time weaving support
  - JBoss-oriented samples
- Historically, a spring-int module inside JBoss AS
- Dedicated forum topic: “JBoss/Spring Integration”

What is Snowdrop? (2)

- Currently, the Spring Deployer and the spring-int jars can be downloaded from Sourceforge
- This will stand as a separate project (Snowdrop)
- Its future home at **http://jboss.org/snowdrop**
- Distinct JIRA project (JBSPRING->SNOWDROP)
- Maven via repository.jboss.org
- Release in September

Spring's support for Java EE integration

- Bootstrapping
- Transaction management
- Messaging support
- Asynchronous tasks
- JPA
- Web Services
- EJB3
- JMX management

Bootstrapping ApplicationContexts in Java EE

- **Best practice**:
  - create the ApplicationContext only once during the lifetime of the application (most common source of trouble!)
- Web applications do it through the ContextLoaderListener
- For EJB (and not only), Spring provides the BeanFactoryLocator abstraction
- For JBoss you have the Spring Deployer

Using BeanFactoryLocator

- Most typical: ContextSingletonBeanFactoryLocator
- By default, loads a group of ApplicationContexts from “classpath*:beanRefContext.xml”, defined like:

```
<beans>
 <bean id="businessBeanFactory" class="o.s.c.s.CPXAC">
   <constructor-arg value="someApplicationContext.xml"/>
 </bean>
</beans>
```

- Classloader-bound singleton (e.g. one per EAR)
- `useBeanFactory(String factoryKey)` retrieves the ApplicationContext identified by the key
- Used internally by Spring for EJB integration

Using Spring Deployer

- JBoss deployer for Spring applications
- Recognizes META-INF/*-spring.xml files (each such file instantiates an ApplicationContext)
- Bootstraps a Spring application context and registers it in the local JNDI (non-serializable)
- Single shared context instance available for all the deployed components
- Advantages:
  - lifecycle is independent of the client application
  - application context is treated as a deployment unit

Common design strategies for bootstrapping

- Most common structure: one ApplicationContext per WAR
  - ContextLoaderListener bootstraps business context
  - DispatcherServlet if using Spring MVC
- If having multiple ApplicationContexts in the same EAR
  - define individual contexts for each web application
  - parent context at the EJB JAR level
    - expensive beans are instantiated only once

A particular problem: VFS support

- JBoss-provided utilities filling a functional gap
- Due to some internal assumptions, Spring's 2.5.x resource/classpath scanning doesn't work properly with JBoss AS's Virtual File System (VFS)
  - Affects resource scanning and annotation-driven configuration
  - Telltale sign:

    `<import resource="classpath*:META-INF/*.xml"/>`, or scanning for @Component-annotated beans

    `<context:component-scan base-package="...."/>`

    will yield:
  - `java.util.zip.ZipException: error in opening zip file`

    `at java.util.zip.ZipFile.open(Native Method)`

VFS Support – The Solution

- The cause of the problem is the behavior of Spring's PathMatchingResourcePatternResolver and the DefaultResourceLoader
- Fortunately, Spring is extensible enough ...
- … so we added two specialized ApplicationContext implementations
  - VFSClassPathXmlApplicationContext
  - VFSXmlWebApplicationContext (for web applications)
    - set the appropriate class through the contextClass parameter in web.xml
- They do all what their parent classes did, plus handling VFS-located resources correctly

VFS Support – Code Samples

- For a web-application bootstrapped servlet

```
<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>
```

- Or apply the same on the DispatcherServlet
- Or use it in a BeanFactoryLocator

Transaction Management with Spring

- Springs allows for **declarative** transaction management (XML and annotation-driven)
- Spring honors @TransactionAttribute
- Delegates to a PlatformTransactionManager, with various implementations
- In Spring 2.5 you can use JTA simply like this:

```
<tx:jta-transaction-manager/ >
```

- The JBoss Transaction Manager will be autodetected
- Synchronize operations with Spring or CMT, using JTA 1.1 support for TransactionSynchronisationRegistry

Spring and JPA

- A common scenario: JPA-based Spring DAOs
- Options:
  - instantiate one of the Local*EntityManagerFactoryBeans
  - retrieve the EntityManagerFactory from JNDI (use <jee:jndi-object />)
  - Inject the EntityManager directly (rather than the EntityManagerFactory) – can be acquired from JNDI
    - In local (non-JNDI) scenarios, use SharedEntityManagerBean

Spring and EJB3

- Some degree of overlapping: both allow implementing application components as POJOs, with middleware services being invoked around the code
- Wrapping Spring beans in EJBs allows for seamlessly integrating with the application-server provided services (transactions, security, management) ...
- … while Spring takes care of injecting dependencies and locating resources (you avoid ServiceLocator)
- Spring beans can be still tested separately
- For testing, you can still use mocks or locally defined EntityManagerFactory/EntityManager

Injecting Spring Beans in EJBs

- Method 1: Spring's SpringBeanAutowiringInterceptor
- EJB3 interceptor
- Uses a BeanFactoryLocator to retrieve the ApplicationContext
- Recognizes the @Autowired annotation

Injecting Spring Beans into EJBs (2)

- Method 2: Based on JBoss' Spring Deployer
- Another EJB3 interceptor: SpringInjectionInterceptor
- Recognizes @Spring annotations
- Injects beans by name
- Requires a JNDI-bound ApplicationContext (normally created by the Spring Deployer)

Spring and Container-Managed Transactions

- Container-managed transactions are one of the benefits of using EJB
- Spring applications can enroll in container-managed transactions
- Use <tx:jta-transaction-manager/> or JtaTransactionManager
- Use managed data sources
  - So that they are enrolled in the corresponding transactions

JPA, CMT and Spring

- JTA challenge: use the same persistence context in EJB and Spring
- Inject the EntityManager acquired from JNDI
  - We said that already, right?
- Do **not** rely on Spring's PersistenceAnnotationBeanPostProcessor
  - It works directly with the EntityManagerFactory
  - Creates a new persistence context
  - It is enrolled in the same transaction, but entities may be loaded twice (conflicts, superfluous operations)

Using PersistenceAnnotationBeanPostProcessor

```
public class JpaUserRepository implements UserRepository {
   @PersistenceContext
   private EntityManager entityManager;
   User findUserByLocation(String place);
}
```

```
<jee:jndi-lookup proxy-interface="javax.persistence.EntityManagerFactory"
    id="entityManagerFactory" jndi-name="java:/persistence/orders-emf"/>
```

```
@Stateless
@Interceptors(SpringInjectionInterceptor.class)
public class UserServiceBean implements UserService
{

   @PersistenceContext(..)
   private EntityManager entityManager;
   ....

   @Spring(jndiName="springContext", bean="userRepository")
   private UserRepository userRepository;
}
```

PersistenceContexts are different !

Injecting the EntityManager

```
public class JpaUserRepository implements UserRepository {
   @Autowired
   private EntityManager entityManager;
   User findUserByLocation(String place);
}
```

```
<jee:jndi-lookup proxy-interface="javax.persistence.EntityManager"
    id="entityManager" jndi-name="java:/persistence/orders"/>
```

```
@Stateless
@Interceptors(SpringInjectionInterceptor.class)
public class UserServiceBean implements UserService
{

   @PersistenceContext(..)
   private EntityManager entityManager;
   ....

   @Spring(jndiName="springContext", bean="userRepository")
   private UserRepository userRepository;
}
```

PersistenceContexts are the same !

Asynchronous Task Execution

- Java EE prohibits the creation of new Threads/ThreadPools by managed components
- Spring allows to delegate the execution of asynchronous tasks to a TaskExecutor instance (Spring provided abstraction)
- For JBoss – use JBossWorkManagerTaskExecutor

  (uses the WorkManager defined for JCA 1.5)
- For example, including a periodically running task as part of your application

Spring in JSF-based web applications

- Spring beans can be used in JSF applications
  - referring to Spring beans in JSF expressions
    - DelegatingVariableResolver
    - SpringBeanVariableResolver
    - SpringBeanFacesELResolver
- Conversational applications may take advantage of Seam-Spring integration
    - Allows injecting Spring beans in Seam beans and vice-versa
    - Allows propagating the Seam-managed persistence context for conversational scenarios

JAX-WS Support

- Spring beans can be exported as Web Services, using JAX-WS support provided by JBoss AS
- Annotate with @WebService
- Define as servlet in web.xml
- Extend from SpringBeanAutowiringSupport
  - supports @Autowired and @Qualifier
- Or export the beans SimpleJaxWsServiceExporter will use JBossWS (on JBoss AS)

Spring-configured JAX-WS Servlets

```
@WebService(serviceName = "UserService", targetNamespace = "")
public class UserWebService extends SpringBeanAutowiringSupport {
    @Autowired UserService userService;

    @WebMethod public boolean exists(String userName) { … }

}
```

```
<bean id="userService" class="springdemo.business.UserServiceImpl">
    <property name="userDao" ref="userDao"/>
</bean>
```

```
<listener>
       <listener-class>o.s.w.c.ContextLoaderListener</listener-class>
</listener>
<servlet>
  <servlet-name>UserWebService</servlet-name>
  <servlet-class>springdemo.ws.UserWebService</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>TestService</servlet-name>
  <url-pattern>/*</url-pattern>
</servlet-mapping>
```

Spring-exported JAX-WS services

```
@WebService(serviceName = "UserService", targetNamespace = "")
public class UserWebService extends SpringBeanAutowiringSupport {
    @Autowired UserService userService;

    @WebMethod public boolean exists(String userName) { … }

}
```

```
<bean id=”userService” class=”springdemo.business.UserServiceImpl”>
    <property name=”userDao” ref=”userDao”/>
</bean>

<bean id=”userWebService” class=”springdemo.ws.UserWebService”/>

<bean class="o.s.remoting.jaxws.SimpleJaxWsServiceExporter">
    <property name="baseAddress" value="http://localhost:8080/myapp" />
</bean>
```

Managing Spring beans with JMX

- Spring beans can be exposed as MBeans
  - typically singletons
- Managed using the JBoss management console
- Parameters can be changed at runtime (turning on/off functionalities, flushing a local cache, etc., collecting information from a monitoring aspect)
- XML and annotation-driven configuration

Exposing Spring Beans as JMX MBeans

```
@ManagedResource("userService:name=UserService")
public class CacheManagerImpl implements CacheManager{

  @ManagedAttribute
  public int getElementCount() {...}

  @ManagedOperation
  public void flush() {...}
  ...
}
```

```
<context:mbean-export/>
```

Future developments for JBoss/Spring utilities

- Better integration between Spring and the Microcontainer
- Improve the Spring Deployer
  - Capitalize on the developments of JBoss MC
  - Add utilities for using JBoss AOP with Spring
  - Add support for standard Spring annotations
- Make suggestions on the forum and report issues in JIRA!

Tooling

- JBoss Tools 3.0 and JBoss Developer Studion 2.0
  - Eclipse-based
- Spring IDE is included
- Configuration validation (bean references, type safety)
- Includes support for annotation-based configuration
- Bean visualization

Red Hat Open Choice Strategy

- Open Choice Strategy: announced on June 1, 2009
- Red Hat's commitment to provide an Open Platform that support popular programming models and deployment paradigms.
- JBoss be the platform of choice to run most popular frameworks
- JBoss users to confidently use their choice of programming model – Seam, GWT, Struts, Spring, RichFaces etc

Red Hat Open Choice Strategy (2)

- Benefits to Red Hat Customers / Developers
  - Single environment for deploying and managing your choice of framework
  - Peace of mind - supported through a trusted vendor
  - Lower overall cost, increased flexibility and ease of development.
- WFK (Web Framework Kit) is a Red Hat product offering based on Open Choice Strategy
- WFK 1.0 is included in JBoss EAP5 and JBoss EWP5 and available for subscription with JBoss EWS 1.0

Spring in WFK

- Enabling JBoss as the preferred platform to develop and deploy Spring applications
- WFK 1.0 includes Spring Framework 2.5.6.SEC01 as a technology preview (among other web frameworks)
- Built and certified by Red Hat
- More to come in future versions:
  - Better integration with JBoss Platforms
  - Will include Snowdrop, the JBoss utilities library
  - Good set of real world samples that leverages the best of Spring and JBoss technologies

Conclusions

- Spring has native capabilities of integrating with JBoss provided services through its Java EE support
- In addition, JBoss-specific utilities for integrating with Spring provide a richer experience
- The JBoss application server and framework landscape provide ample opportunities to run Spring applications efficiently
- JBoss products such as WFK for providing customer support

QUESTIONS?

TELL US WHAT YOU THINK:
REDHAT.COM/JBOSSWORLD-SURVEY