Merging EJB3 and Spring frameworks

Aleš Justin
Genera Lynx d.o.o.

Overall agenda

- EJB3 dependency injection
- Spring POJOs
- JBoss AS deployers
- Injecting Spring into EJBs
- Portability with EJB3 interceptors
- Use case – JBoss Portal app
- JBoss Spring and Microcontainer

Goals of EJB 3.0 (by Bill Burke from JBW Barcelona)

- EJB 2.1 is too noisy
  - ✓ Too many interfaces to implement
  - ✓ “XML Hell” too many complex deployment descriptors
  - ✓ API is too verbose and complicated
- Simplify the EJB programming model
- Focus on ease of use
- Facilitate Test Driven Development
- Make it simpler for average developer
- Increase developer base

EJB3 dependency injection

Injection – how grained?

- EJB3 coarse-grained injection
  - ✓ @EJB, @Resource, @PersistenceContext
  - ✓ @JndiInject
  - ✓ XML alternative to injection annotations

```java
@Stateful
public class ShoppingCartBean implements ShoppingCart {

   @Resource(name="DefaultDS") private DataSource ds;
   @EJB private CreditCardProcessor ccp;

   @JndiInject(jndiName = "java:portal/SessionFactory")
   private SessionFactory portalSessionFactory;

   private EntityManager em;

   @PersistenceContext(unitName="orderdb")
   public void setEntityManager(EntityManager em) {
     this.em = em;
   }
```

```xml
<ejb-ref>
   <ejb-ref-name>ejb/CreditCardProcessor</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <local>com.acme.CreditCardProcessor</local>
   <ejb-link>CreditCardEJB</ejb-link>
   <inject-target>ccp</inject-target>
</ejb-ref>
```

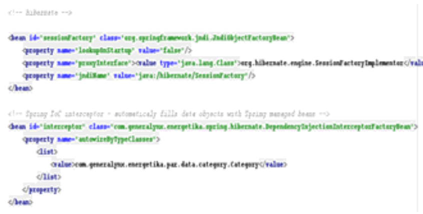
Injection – hitting the wall

- can only inject Java EE component types
- <env-entry> only lets you inject primitive types
- beans from third-party libraries
- pre-existing code
- instantiation and construction

Spring POJOs

POJOs – how grained?

- Spring fine-grained beans
  - ✓ beans from third-party libraries
  - ✓ pre-existing code
  - ✓ instantiation and construction



```
<!-- hibernate -->

<bean id='sessionFactory' class='org.springframework.jndi.JndiObjectFactoryBean'>
    <property name='lookupOnStartup' value='false'/>
    <property name='proxyInterface'><value type='java.lang.Class'>org.hibernate.engine.SessionFactoryImplementor</value></property>
    <property name='jndiName' value='java:/hibernate/SessionFactory'/>
</bean>

<!-- Spring IoC interceptor - automaticaly fills data objects with Spring managed beans -->
<bean id='interceptor' class='com.generalynx.energetika.spring.hibernate.DependencyInjectionInterceptorFactoryBean'>
    <property name='autowireByTypeClasses'>
        <list>
            <value>com.generalynx.energetika.par.data.category.Category</value>
        </list>
    </property>
</bean>
```

Why Spring?

- good experience from previous application development
- useful utility classes
  - ✓ property configurability
  - ✓ lazy initialization
  - ✓ JNDI lookup
  - ✓ JMX exporting
  - ✓ 3rd party library integration
- new XML configuration (from 2.0)

More Spring POJO power

- lifecycle support
  - ✓ interfaces
  - ✓ method descriptions
- plain Bean Factory
- full Application Context
  - ✓ post processors
    - BeanFactory
    - Bean
  - ✓ message source
  - ✓ event multicaster

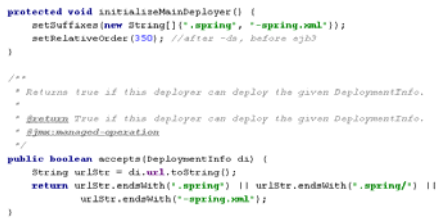
JBoss AS deployers

Deployable units

- already familiar / using deployers
  - ✓ .ear, .war, -ds.xml, .har, …
- architecture
  - ✓ Scanner
  - ✓ JAR Archive, XMLs, exploded
  - ✓ classloader creation
  - ✓ deployment order (Deployers are also deployable units)
  - ✓ JMX MBeans

Integration / extension

- simple – abstract API
  - ✓ org.jboss.deployment.SubDeployerSupport
  - ✓ defining your components



```java
protected void initialiseMainDeployer() {
    setSuffixes(new String[]{".spring", "-spring.xml"});
    setRelativeOrder(350); //after -ds, before ejb3
}

/**
 * Returns true if this deployer can deploy the given DeploymentInfo.
 *
 * @return True if this deployer can deploy the given DeploymentInfo.
 * @jmx:managed-operation
 */
public boolean accepts(DeploymentInfo di) {
    String urlStr = di.url.toString();
    return urlStr.endsWith(".spring") || urlStr.endsWith(".spring/") ||
           urlStr.endsWith("-spring.xml");
}
```

SpringDeployer

- supports
  - ✓ raw Spring XML description file
  - ✓ .spring JAR archive
  - ✓ exploded .spring archive

SpringDeployer

- lifecycle
  - ✓ create bean factory / application context
  - ✓ global scope classloader
  - ✓ register under JNDI
    - default short name – file name
    - BeanFactory=(<name>)
  - ✓ destroy
  - ✓ unregister

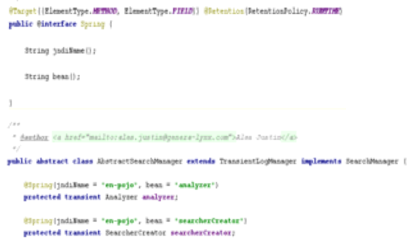
SpringDeployer

- parent Bean Factory / App. Context
  - ✓ ParentBeanFactory=(<name>)
  - ✓ hierarchy / order
    - conf/jboss-service.xml
    - URLDeploymentScanner
    - org.jboss.deployment.DeploymentSorter

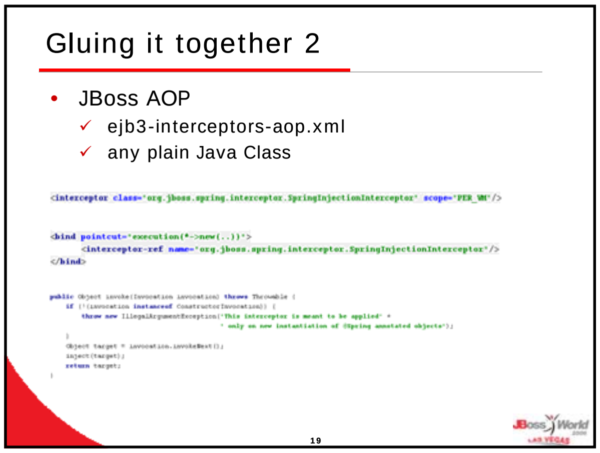
Injecting Spring into EJBs

Gluing it together

- non-serializable JNDI binding
- annotations



```java
@Target({ElementType.METHOD, ElementType.FIELD}) @Retention(RetentionPolicy.RUNTIME)
public @interface Spring {

    String jndiName();

    String bean();

}

/**
 * @author <a href="mailto:ales.justin@genera-lynx.com">Ales Justin</a>
 */
public abstract class AbstractSearchManager extends TransientLogManager implements SearchManager {

    @Spring(jndiName = "en-pojo", bean = "analyzer")
    protected transient Analyzer analyzer;

    @Spring(jndiName = "en-pojo", bean = "searcherCreator")
    protected transient SearcherCreator searcherCreator;
```

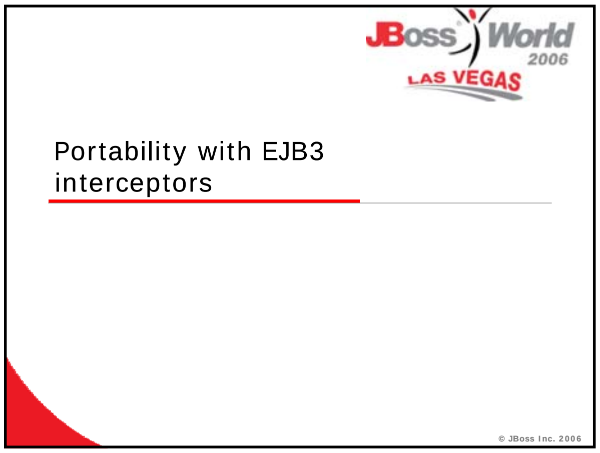


Gluing it together 2

- JBoss AOP
  - ✓ ejb3-interceptors-aop.xml
  - ✓ any plain Java Class

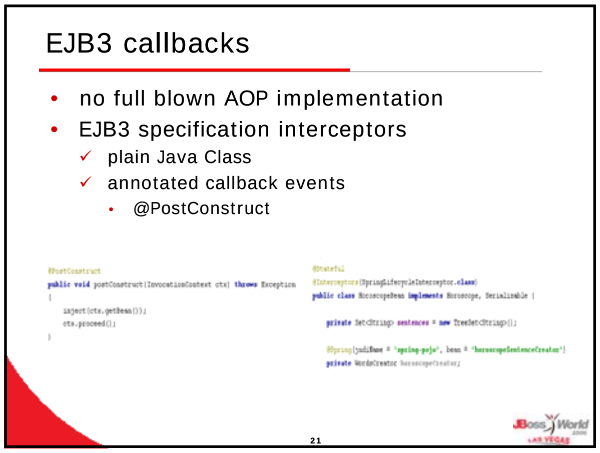
```
<interceptor class="org.jboss.spring.interceptor.SpringInjectionInterceptor" scope="PER_VM"/>

<bind pointcut="execution(*->new(..))">
      <interceptor-ref name="org.jboss.spring.interceptor.SpringInjectionInterceptor"/>
</bind>

public Object invoke(Invocation invocation) throws Throwable {
    if (!(invocation instanceof ConstructorInvocation)) {
        throw new IllegalArgumentException("This interceptor is meant to be applied" +
                                           " only on new instantiation of @Spring annotated objects");
    }
    Object target = invocation.invokeNext();
    inject(target);
    return target;
}
```

19



Portability with EJB3 interceptors

© JBoss Inc. 2006



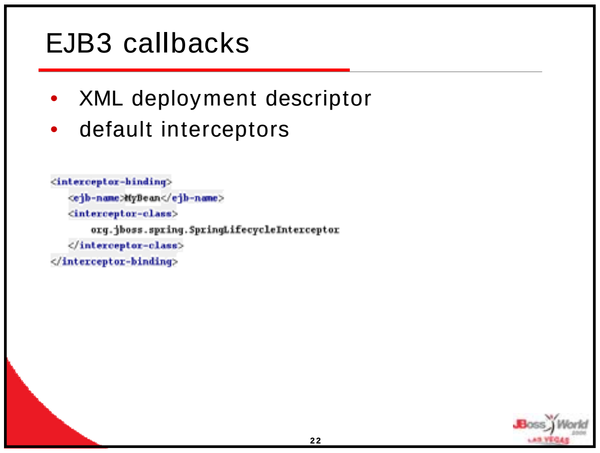
EJB3 callbacks

- no full blown AOP implementation
- EJB3 specification interceptors
  - ✓ plain Java Class
  - ✓ annotated callback events
    - @PostConstruct

```
@PostConstruct
public void postConstruct(InvocationContext ctx) throws Exception
{
    inject(ctx.getBean());
    ctx.proceed();
}
```

```
@Stateful
@Interceptors(SpringLifecycleInterceptor.class)
public class HoroscopeBean implements Horoscope, Serializable {

    private Set<String> sentences = new TreeSet<String>();

    @Spring(jndiName = "spring-pojo", bean = "horoscopeSentenceCreator")
    private WordsCreator horoscopeCreator;
```

21

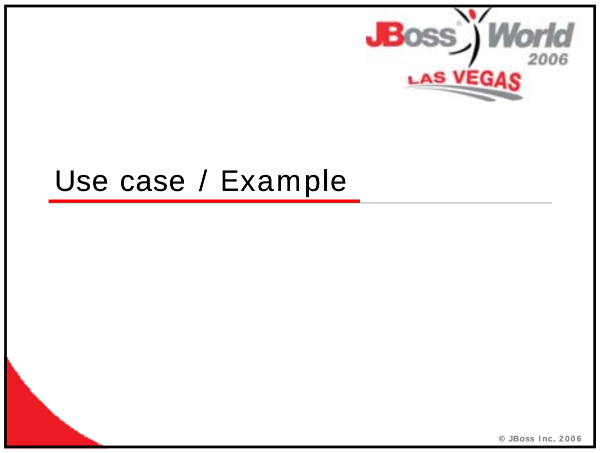


EJB3 callbacks

- XML deployment descriptor
- default interceptors

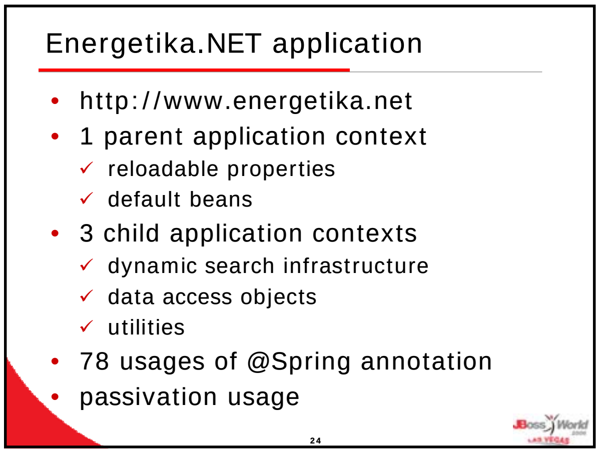
```
<interceptor-binding>
   <ejb-name>MyBean</ejb-name>
   <interceptor-class>
      org.jboss.spring.SpringLifecycleInterceptor
   </interceptor-class>
</interceptor-binding>
```

22



Use case / Example

© JBoss Inc. 2006



Energetika.NET application

- http://www.energetika.net
- 1 parent application context
  - ✓ reloadable properties
  - ✓ default beans
- 3 child application contexts
  - ✓ dynamic search infrastructure
  - ✓ data access objects
  - ✓ utilities
- 78 usages of @Spring annotation
- passivation usage

24

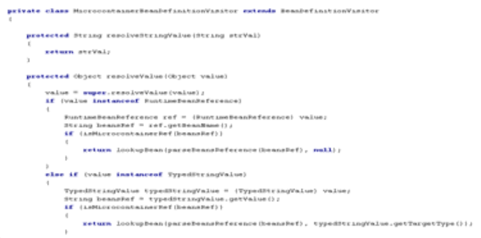
JBoss Spring and Microcontainer

JBoss Microcontainer

- replacing JMX based kernel
- plain POJO kernel
- extreme lifecycle support
- ‘powers’ embeddable EJB3

Spring → Microcontainer

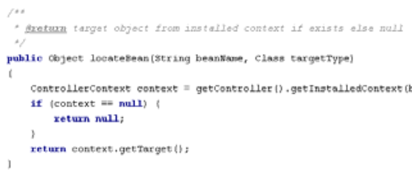
- non-invasive
  - ✓ used as post processor
  - ✓ SpringMicrocontainerDeployer



```
private class MicrocontainerBeanDefinitionVisitor extends BeanDefinitionVisitor
{

   protected String resolveStringValue(String strVal)
   {
      return strVal;
   }

   protected Object resolveValue(Object value)
   {
      value = super.resolveValue(value);
      if (value instanceof RuntimeBeanReference)
      {
         RuntimeBeanReference ref = (RuntimeBeanReference) value;
         String beansRef = ref.getBeanName();
         if (isMicrocontainerRef(beansRef))
         {
            return lookupBean(parseBeansReference(beansRef), null);
         }
      }
      else if (value instanceof TypedStringValue)
      {
         TypedStringValue typedStringValue = (TypedStringValue) value;
         String beansRef = typedStringValue.getValue();
         if (isMicrocontainerRef(beansRef))
         {
            return lookupBean(parseBeansReference(beansRef), typedStringValue.getTargetType());
         }
```

Locator class

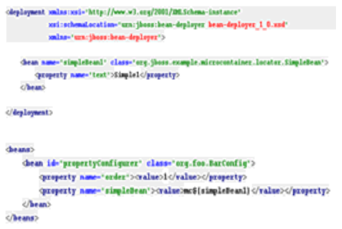
- locating Kernel instance
- locating KernelController instance
- yet to implement



```
/**
 * @return target object from installed context if exists else null
 */
public Object locateBean(String beanName, Class targetType)
{
   ControllerContext context = getController().getInstalledContext(beanName);
   if (context == null) {
      return null;
   }
   return context.getTarget();
}
```

Usage

- currently only one way
  - ✓ from Spring locating already instantiated Microcontainer POJOs



```
<deployment xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="urn:jboss:bean-deployer bean-deployer_1_0.xsd"
            xmlns="urn:jboss:bean-deployer">

   <bean name="simpleBean1" class="org.jboss.example.microcontainer.locator.SimpleBean">
      <property name="text">Simple1</property>
   </bean>

</deployment>


<beans>
    <bean id="propertyConfigurer" class="org.foo.BarConfig">
        <property name="order"><value>1</value></property>
        <property name="simpleBean"><value>mc${simpleBean1}</value></property>
    </bean>
</beans>
```

JBoss Spring module

- download at http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=161914
- supports Spring framework 2.0m3
  - ✓ previous releases based on 1.2.4
- forum support
  - ✓ http://www.jboss.org/index.html?module=bb&op=viewforum&f=223