



J-Spring

16 april 2008 Spant! - Bussum



Using Spring 2.5 with Java EE 5

Joris Kuipers





Spring & Java EE 5

- Short history
- Spring 2.x and J2EE integration
- New possibilities in Spring 2.5
- Spring 2.5 & EJB 3.0
 - Integration
 - Comparison
- Conclusion and Q&A



Short J2EE / Java EE history

- Started in 1998
 - Umbrella specification from different vendors
- Newer versions have both updated and more specs
 - e.g., addition of web services functionality
 - 1.2: Dec. '99, 1.3: Sep. '01, 1.4: Nov. '03, 1.5: May '06
- Quickly became popular, despite some deficiencies
- Market currently dominated by handful of players
- Java EE 6 is in the works



Short Spring history

- Started end 2002 from Expert J2EE One-on-One
 - Influential book by Rod Johnson
 - Openly criticized EJB
- Code became Open Source project
 - Lightweight Application Framework (not just J2EE)
 - Reached 1.0 in March 2004
 - 1.1: Sep. '04, 1.2: May '05, 2.0: Oct. '06, 2.5: Nov. '07
- Interface21 founded in July 2004
 - Became SpringSource in Nov. '07
- Currently has many related projects



Spring & J2EE integration

- Spring has integrated with J2EE technologies since its inception
- Typically doesn't reinvent the wheel
 - Unless the current wheel is square 😊
- JNDI, JTA, JPA, JMS, JSF, EJB, etc.
- XML namespaces for many of these



Integration Examples

- `<jee:jndi-lookup id="dataSource" jndi-name="jdbc/myDS" resource-ref="true"/>`
 - supports caching, lazy-lookup, and custom environment
- `<tx:jta-transaction-manager/>`
 - generic JTA as well as vendor-specific extensions
- `<jee:local-slsb id="myBean" jndi-name="ejb/myBean" business-interface="com.mycom.MyComponent"/>`
- LocalContainerEntityManagerFactoryBean
 - Let Spring act as JPA-container without EJB-container



JMS Integration

- Particular Spring strongpoint
- Sending and receiving
- "MDB's on steroids"
 - In any environment
 - Standard MessageListeners as well as POJO's
 - Pluggable thread pool through TaskExecutor
 - Supports request/response model



JMS Integration Examples

```
<jms:listener-container>
  <jms:listener destination="queue.confirmations"
    ref="myConfirmationListener"/>
  <jms:listener destination="queue.orders"
    ref="orderService" method="placeOrder"/>
</jms:listener-container>
```

- Default container
- JMS/Spring MessageListener on confirmations
- POJO listener on orders
 - Supports `placeOrder(String)`, `placeOrder(byte[])`, etc.



JSF Integration

- Spring beans accessed from or acting as JSF backing beans
- JSF 1.1: DelegatingVariableResolver
JSF 1.2: SpringBeanFacesELResolver
- Much more powerful
 - Full Dependency Injection
 - AOP
 - etc.



JSF Integration

- Web Flow 2 takes JSF integration to the next level
- Separate component: **spring-faces**
 - Client-side validation
 - AJAX with JSF
 - Progressive UICommand components
- Can also use JSF and Facelets as pure view technology in Spring-MVC



What's new in Spring 2.5

Also for stand-alone usage!

- New Java EE 5 API's supported:
 - Servlet 2.5, JSP 2.1 & JSF 1.2
 - JTA 1.1, JAX-WS 2.0 & JavaMail 1.4
- Support for JSR-250 Common Annotations
- Custom namespaces for JMS and JCA
- Inject Spring Beans in EJB 3 Session- or Message-driven Beans



JSR-250 Common Annotations

- Part of Java EE 5 and Java SE 6
- Fully supported by Spring 2.5!
- Enabled through custom BeanPostProcessor
- Or with single element:
`<context:annotation-config/>`





@Resource

- Dependency Injection by name
 - Java EE: JNDI name
 - Spring: Bean name or JNDI name

```
public class MyService implements MyServiceInterface {  
    @Resource  
    private DataSource dataSource;  
  
    private Processor processor;  
  
    @Resource(name="myProcessor")  
    public void setProcessor(Processor processor) { ... }  
  
    ...  
}
```



@Resource

- Spring also has its own @Autowired
 - Dependency Injection by Type
 - More powerful, but Spring-specific
- Nice when Spring-dependency is not an issue
 - e.g. Spring-MVC Controllers
 - Or when you just don't care 😊



@PostConstruct / @PreDestroy

- Lifecycle callbacks
 - Act as Spring's init- and destroy-method
 - Or InitializingBean and DisposableBean
 - Work exactly like in a Java EE container
- Benefits:
 - Not Spring-specific
 - Cannot forget to add to external configuration



Lifecycle callbacks example

```
public class MyService implements MyServiceInterface {
    @Resource private DataSource dataSource;
    @Resource private Processor processor;

    @PostConstruct
    public void initService() {
        // you can safely use dataSource and processor here
    }

    @PreDestroy
    public void cleanUpAfterOurselves() {
        // called when object is 'destroyed' (i.e., application closes)
    }
}
```



Further Java EE 5 Annotations

- **@WebServiceRef / @EJB**
 - injecting a JAX-WS / EJB 3 service proxy
- **@TransactionAttribute**
 - EJB 3 transaction demarcation
- **@PersistenceContext / @PersistenceUnit**
 - JPA resource injection
 - supported since Spring 2.0 already!
- All consistently supported



Conclusion

- Spring is **not** 'anti-Enterprise Java' or 'anti-standards'
 - Integrates with all of the Java EE standard
 - Natively supports Common Annotations
- Doesn't make you swallow the whole elephant: use what *you* need
 - In many cases just Tomcat
 - Or a full-blown application server
- Spring is all about **choice**



Spring 2.5 and EJB 3.0

Integration & Comparison



Overview

- EJB 3.0 has some new features similar to Spring
 - POJO-based programming model
 - Limited Dependency Injection Support
- Spring 2.5 added Annotation-based configuration support
- How do they compare and integrate?



Integrating Spring 2.5 & EJB 3.0: Using EJB from Spring

- EJB 3.0 requires just a JNDI lookup
 - No more `BeanHome.create()`
- So use `<jee:jndi-lookup/>`
- Caching, lazy-loading, etc. all built-in
- Spring can manage security and tx's
- Useful to communicate with local or remote stateless session EJB's



Integrating Spring 2.5 & EJB 3.0: Using Spring from EJB

- Obtain `ApplicationContext` and perform lookup using `getBean()`
 - Also available on EJB 2.x
- Or use **SpringBeanAutowiringInterceptor**
 - EJB 3.0 Interceptor that applies autowiring: Spring DI of EJB 3.0 bean!
 - No need to obtain ref to `ApplicationContext` and do a lookup yourself
- Let's look at a demo...



Other integration option

- **Spring Pitchfork**
- EJB 3.0 container based on Spring
 - Without JPA
- Every EJB *is* a Spring bean
- Basis for WebLogic
- Not widely used outside WebLogic
 - Most people just use Spring



Comparison

- Integration is possible, but not typical
- EJB 3.0 or Spring is rather either/or choice for most apps
- You know what we prefer...
- But why? EJB 3.0 is a lot better than EJB 2.1, right?
- Let's compare some things
 - From our perspective...



Topics that can be compared

- Backwards compatibility
- Pace of innovation
- Dependency Injection capabilities
- Applicability in environments
- Dealing with cross-cutting concerns
- Scopes and pooling
- Transaction management capabilities



Backwards compatibility: EJB

- EJB 3.0 includes support for 2.x and 1.x
 - Can still deploy your old apps
- But programming model is *completely* different between major releases
- No real upgrade path
 - just rebuild your app if you want to use new features and improvements
- EJB 3.0 requires new JDK and new application server



Backwards compatibility: Spring

- Spring 2.0 and 2.5 almost 100% backwards compatible => drop-in replacement
- With the *same* programming model
- Most apps don't even have compile time dependency
 - apart from library classes like templates
- No forced upgrade of JDK or app server
 - but supports new features if present



Pace of innovation

- EJB is part of Java EE spec
 - Moves slow because of many parties involved
 - Implementations take even longer
 - Results in 'too little, too late'
- Spring *uses* Java EE, but doesn't *depend* on it
 - Can follow its own release schedule
 - No need to buy a new application server



Dependency Injection capabilities: EJB 3.0

- Pretty much just Annotations and JNDI
 - Values through very verbose XML
 - No plain objects
- Needless coupling
 - @EJB? @WebServiceRef? It's just a service!
- Limited
 - No constructors, arbitrary methods for DI
 - No factory methods for construction
 - Per instance configuration is hard



Dependency Injection capabilities: Spring 2.5

- XML and Annotations both 1st class citizens
- Can inject everything into everything
 - Not just from JNDI
 - Not just into Java EE objects
- Generic
 - @Autowired / @Resource for everything
- Advanced DI capabilities
 - Constructors, fields, arbitrary methods
 - Factory methods, lookups, etc.
 - Easy to do per instance in XML



Applicability in environments

- EJB 3.0 requires Java EE 5 Container
 - Limits applicability
 - Limits reusability of your code
 - Harder to do integration testing
- Spring can be used everywhere
 - Standalone, Servlet-only, Java EE 5, OSGi
 - Allows same code to be used
 - Still integrates with available features
 - Facilitates local integration testing



Dealing with cross-cutting concerns

- EJB 3.0 introduces Interceptors
 - "poor man's AOP": no *pointcut model*
 - Only offers 'around'
- Spring AOP provides real AOP
 - Limited because of proxy-based approach
- Spring integrates with AspectJ for full-blown AOP
 - You can do anything you need
 - Spring 2.5 enables load time weaving



Scopes and pooling

- EJB's are always pooled
 - Because object creation is slow, right?
 - Singleton planned for 3.1
- Have no support for scoping
 - Addressed by upcoming Web Beans spec
- Spring has different scopes since 2.0
 - Including request and session for web
- Spring supports pooling of objects
 - But it's hardly ever needed



TX management capabilities

- TX mgt has always been EJB's strongpoint
- But it's tied to JTA
 - No local TX's, no nested TX's, extra overhead
- Declarative only through EJB CMT
 - Fixed rollback rule
 - RuntimeException -> rollback, misc -> commit
- Spring makes JTA an option, not a requirement
 - Use local TX's if you don't need 2PC
 - Faster and more portable
- More flexible
 - EJB + rollback rules, TransactionTemplate



Conclusion

- Java EE > EJB
- EJB 3.0 big step forward from 2.1, but still a long way from Spring in most areas
- Standards should deliver real value
 - Not 'standards for standards sake'
- Spring is actively involved with Java (EE)
 - JMX, JSF, Java EE 6, Validation, etc.
- Choice is up to you



Questions?