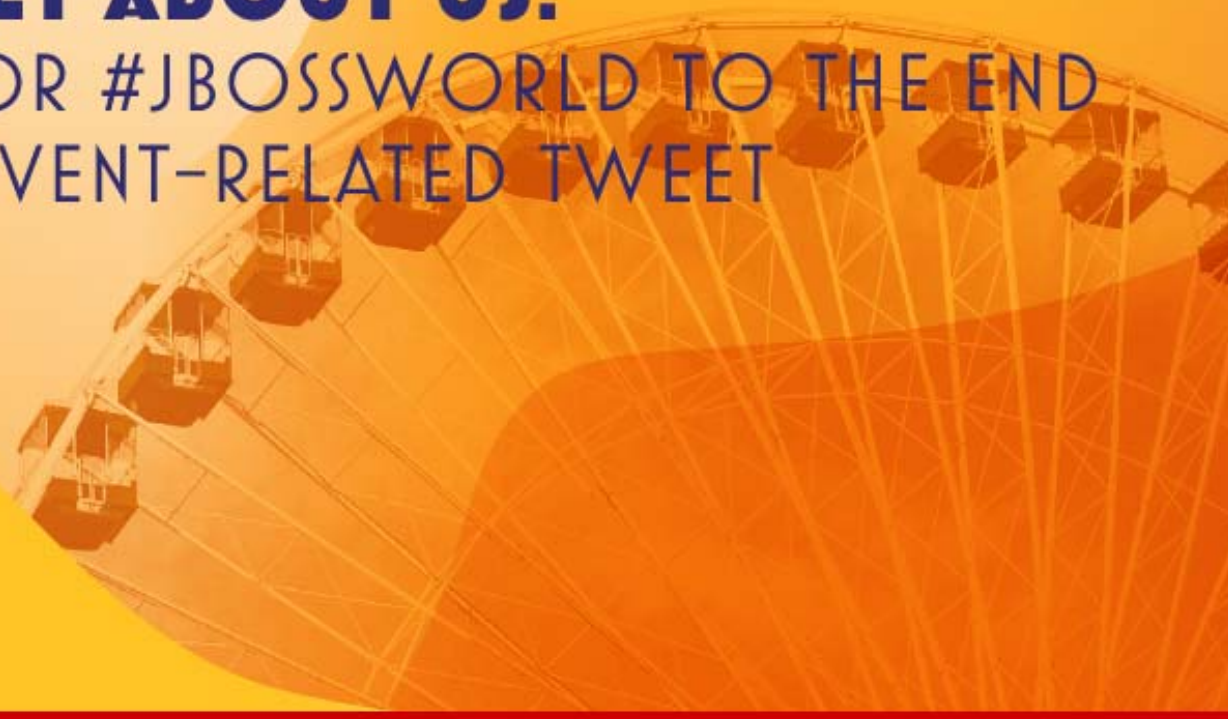
JBoss WORLD
CHICAGO 2009

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

Java EE and the Spring Framework: Compare/Contrast

Reza Rahman
Independent Consultant
September 2009

presented by

A Quick Glance Back

- Java EE pioneers most innovations in server-side Java
- A widely deployed but much maligned platform
- Spring, along with Hibernate, brings POJO programming, IoC and testability to the mainstream
- Java EE 5 adopts ideas, as well as annotations and convention-over-configuration
- Hibernate standardized into JPA, Spring remains non-standard but adopts Java EE features to a degree
- Java EE 6 matures ideas in Java EE 5 in addition to pruning and profiles
- CDI, JSF 2, JPA 2, EJB 3.1, Servlet 3, JAX-RS major API changes

The Birds-Eye View

- Java EE and Spring are both capable middleware solutions
- Functionally equivalent for the most part, with very different approaches
- For very vanilla applications, code is almost identical

Java EE

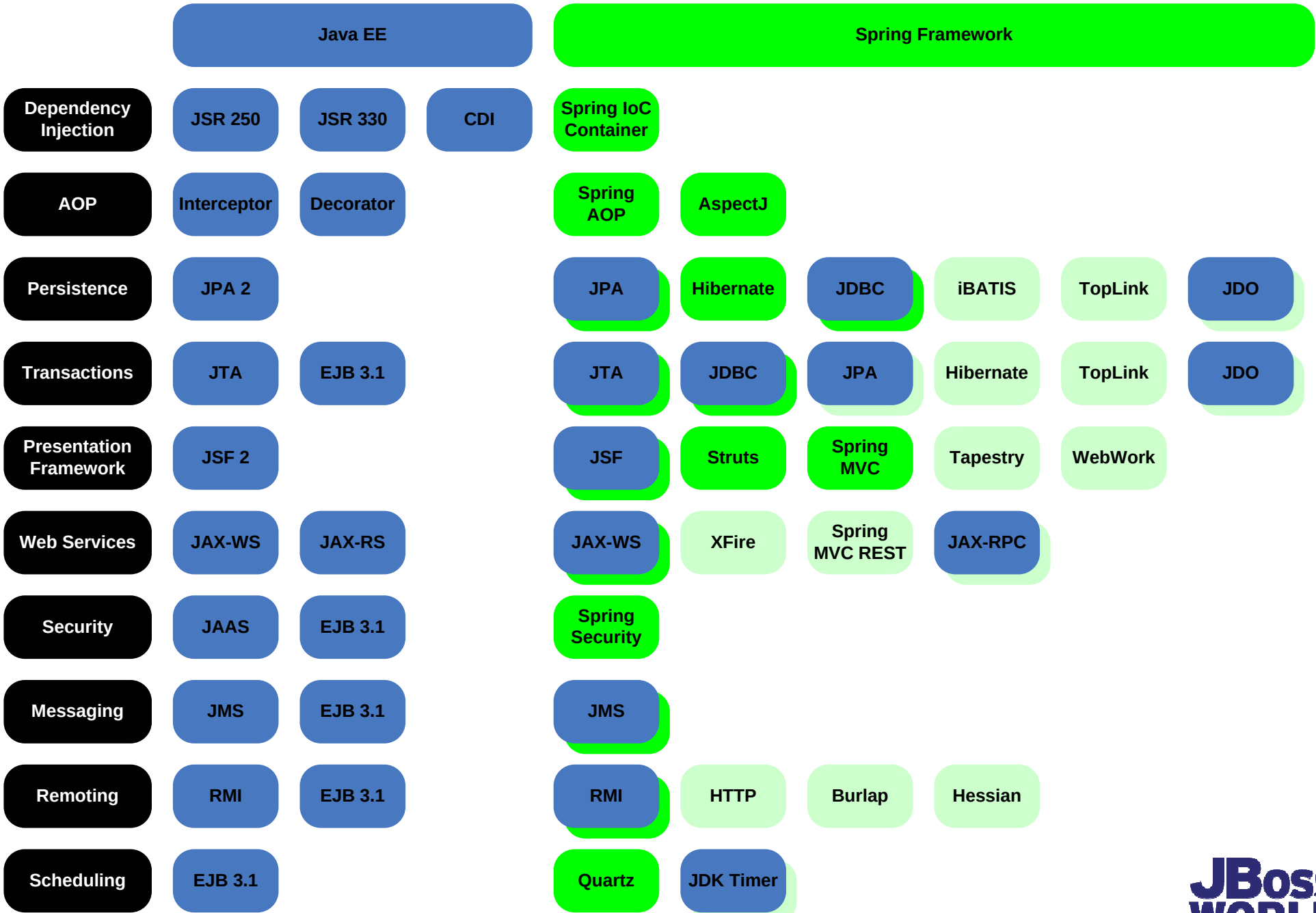
- Fully integrated platform with strong intelligent defaulting and minimal configuration (convention-over-configuration)
- Non-redundant APIs with specialized roles
- Focus on annotations for meta-data and type-safety
- Widely supported open standard
- Vendor and platform agnostic code
- Integration with non-standard technologies left to vendors
- Vendors free to innovate via non-standard extensions and value-added features like clustering, HA, administration, monitoring, development/deployment tools

Spring Framework

- Explicit configuration, typically through XML
- More flexibility for fine-grained control but more complexity
- Ease-of-use abstractions over lower-level, general purpose APIs like JDBC, JMS and JavaMail
- Non-standard open source technology from a commercial vendor
- Integration with standard as well as non-standard technologies that are often overlapping
- Portability across runtime platforms at the risk of vendor lock-in

- There is no practical reason to be forced to choose between the two
- Balanced competition in server-side Java is good for both developers and vendors
- There are excellent integration possibilities

Features/APIs Overview



Java EE Basic Business Component

```
@Stateless
public class BidService {
  @PersistenceContext
  private EntityManager entityManager;

  public void addBid(Bid bid) {
    entityManager.persist(bid);
  }
}
```

The bean is thread-safe; using non thread-safe resources requires no special precautions

All business methods are transactional by default.

Spring Basic Business Component

```
@Service
public class BidService {
  @PersistenceContext
  private EntityManager entityManager;

  @Transactional
  public void addBid(Bid bid) {
    entityManager.persist(bid);
  }
}
```

Bean has no thread-safety guarantees, all resources must be thread-safe.

Spring specific data access APIs handle thread-safe behind scenes, like non-standard entity manager proxy.

Transactions must be specified explicitly

Spring Basic Business Component Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
  <context:component-scan base-package="com.actionbazaar"/>
  <tx:annotation-driven/>
  ...
  <bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
  </bean>
  <bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="loadTimeWeaver">
      <bean
          class="org.springframework.instrument.classloading.InstrumentationLoadTimeWeaver"/>
    </property>
  </bean>
</beans>
```

Schemas reduce configuration burden, but still can be hard to understand, debug, maintain

Annotation driven configuration helps greatly if you are not a fan of XML

Explicit, fine-grained container configuration

May or may not be needed depending on the JPA provider.

Java EE Interceptor

```
@Stateless
public class BidService {
  @Legacy private BidDao bidDao;

  @Audited
  public void addBid(Bid bid) {
    bidDao.addBid(bid);
  }
}

@Interceptor @Audited
public class AuditInterceptor {
  @AroundInvoke
  public Object audit(InvocationContext context) {
    System.out.println("Executing: "
      + context.getMethod().getName());
    return context.proceed();
  }
}

@InterceptorBindingType
@Target({TYPE, METHOD})
@Retention(RUNTIME)
public @interface Audited {}
```

Annotated method intercepted

Interceptor to handle the annotation

Binds the annotation to an interceptor

Spring Aspects

```
@Service
public class BidService {
  @Autowired @Legacy private BidDao bidDao;

  @Audited
  @Transactional
  public void addBid(Bid bid) {
    bidDao.addBid(bid);
  }
}

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Audited {}

@Component
@Aspect
public class AuditAspect {
  @Before("execution(public * *(..)) && @annotation(Audited)")
  public void audit(JoinPoint joinPoint) {
    System.out.println("Entering: " + joinPoint);
  }
}
```

The annotation by itself does not mean anything

More fine-grained interception, but more complex syntax

Spring Aspects Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
  ...
  <aop:aspectj-autoproxy/>
  ...
</beans>
```

Enables @AspectJ, but aspects themselves must still be registered as beans

Java EE Injection

```
@Stateless
public class BidService {
  @Legacy private BidDao bidDao;
  ...
}

@Legacy @Dao
public class NativeSqlBidDao implements BidDao {
  ...
}

@ApplicationScoped
@Profiled
@CurrencyConverted(Currency.USD)
@Stereotype
@Target(TYPE)
@Retention(RUNTIME)
public @interface Dao {}

@BindingType
@Retention(RUNTIME)
@Target({METHOD, FIELD, PARAMETER, TYPE})
public @interface Legacy {}
```

Qualified injection

Grouping metadata via stereotypes

Spring Injection

```
@ActionBazaarService
public class BidService {
  @Autowired @Legacy private BidDao bidDao;
  ...
}

@Legacy @Repository
public class NativeSqlBidDao implements BidDao {
  ...
}

@Qualifier
@Retention(RUNTIME)
@Target({METHOD, FIELD, PARAMETER, TYPE})
public @interface Legacy {}
```

Spring stereotype concept simply loads a component to the registry.

Qualified injection

Spring Injection Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
  ...
  <context:component-scan base-package="com.actionbazaar">
    <context:include-filter type="annotation"
        expression="com.actionbazaar.ActionBazaarService"/>
  </context:component-scan>
  ...
</beans>
```

Including the new component type

Facelet

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:bz="http://java.sun.com/jsf/composite/actionbazaar">
<h:head>
  <title>Add Bid</title>
</h:head>
<h:body>
  <h:form>
    <h3>Add Bid</h3>
    <bz:bidPanel id="bidPanel">
      <f:actionListener for="bidEvent"
        binding="#{bidManager.addBid}" />
    </bz:bidPanel>
  </h:form>
</h:body>
</html>
```

Re-usable custom Facelet component

EL binding expression to beans

Facelet Component

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:composite="http://java.sun.com/jsf/composite">
<body>
  <composite:interface>
    <composite:actionSource name="bidEvent"/>
  </composite:interface>
  <composite:implementation>
    <p>Item: <h:outputText value="#{item.name}"/></p>
    <p>Current bid: <h:outputText
                      value="#{item.highestBid.amount}"/></p>
    <p>Amount: <h:inputText id="amount"
                 value="#{bid.amount}"/></p>
    <p><h:commandButton id="bidEvent" value="Add bid"/>
  </composite:implementation>
</body>
</html>
```

Component interface

Component implementation

EL binding expression to beans

Entity

```
@Named
@Entity
@Table(name="BIDS")
public class Bid {
  @Id
  @GeneratedValue
  private long id;

  @ManyToOne
  @NotNull
  private User bidder;

  @ManyToOne
  @NotNull
  private Item item;

  @Min(1)
  private double amount;

  ...
}
```

Assigns name used for EL binding.

Bean validation constraints

JSF Event Handler

```
@Named
@RequestScoped
public class BidManager {
  @Current private BidService bidService;
  @LoggedIn private User user;
  @SelectedItem private Item item;
  @Current private Bid bid;

  public String addBid() {
    bid.setBidder(user);
    bid.setItem(item);
    bidService.addBid(bid);

    return “bid_confirm.xhtml”;
  }
}
```

Qualified injection

Handling event raised by component

Spring MVC JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html>
<head>
  <title>Add Bid</title>
</head>
<body>
  <form:form modelAttribute="bid" method="post">
    <h3>Add Bid</h3>
    <p>Item: <form:errors path="item" cssClass="error"/>
             ${item.name}</p>
    <p>Current bid: ${item.highestBid.amount}</p>
    <p>Amount: <form:errors path="amount" cssClass="error"/>
               <form:input path="amount"/></p>
    <p><input type="submit" value="Add Bid">
  </form:form>
</body>
</html>
```

Entity

```
@Entity
@Table(name="BIDS")
public class Bid {
  @Id
  @GeneratedValue
  private long id;

  @ManyToOne
  private User bidder;

  @ManyToOne
  private Item item;

  private double amount;

  ...
}
```

Spring Validator

```
public class BidValidator implements Validator {
  public boolean supports(Class clazz) {
    return Bid.class.equals(clazz);
  }

  public void validate(Object object, Errors errors) {
    Bid bid = (Bid) object;

    if (bid.getBidder() == null) {
      errors.rejectValue("bidder", null, "Bidder must not be empty");
    }

    if (bid.getItem() == null) {
      errors.rejectValue("item", null, "Item must not be empty");
    }

    if (bid.getAmount() < 1) {
      errors.rejectValue("amount", null,
          "Amount must be greater than one");
    }
  }
}
```

Spring Controller

```
@Controller
@RequestMapping("/add_bid.do")
@SessionAttributes("item")
@SessionAttributes("bid")
public class BidController {
  @Autowire private ItemService itemService;
  @Autowire private BidService bidService;
  @Autowire private BidValidator bidValidator;

  @RequestMapping(method=RequestMethod.GET)
  public String setupForm(@RequestParam("itemId") int itemId, ModelMap model) {
    Item item = itemService.getItem(itemId);
    model.addAttribute("item", item);
    Bid bid = new Bid();
    model.addAttribute(“bid”, bid);

    return "add_bid";
  }

  @RequestMapping(method=RequestMethod.POST)
  public String addBid(@ModelAttribute("item") Item item, @ModelAttribute("bid") Bid bid,
      HttpSession session, BindingResult result, SessionStatus status) {
    bid.setBidder((User)session.getAttribute(“user”));
    bid.setItem(item);
    bidValidator.validate(bid, result);

    if (result.hasErrors()) {
      return "add_bid";
    } else {
      bidService.addBid(bid);
      status.setComplete();
      return "redirect:confirm_bid.do?itemId=" + item.getItemId();
    }
  }
}
```

Spring MVC Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-2.5.xsd">
  <context:component-scan base-package="com.actionbazaar"/>
  <bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/jsp/"/>
    <property name="suffix" value=".jsp"/>
  </bean>
</beans>
```

Spring MVC web.xml Configuration

```
<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath*:**/applicationContext*.xml
    </param-value>
  </context-param>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>actionbazaar</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>actionbazaar</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>jsp/index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Java EE Remoting and Web Services

```
@Stateless
public class DefaultBidService implements BidService {
  @PersistenceContext
  private EntityManager entityManager;

  public void addBid(Bid bid) {
    entityManager.persist(bid);
  }
}

@Remote
@WebService
public interface BidService {
  void addBid(Bid bid);
}
```

No further configuration needed; A JAX-RS (REST) end-point could have been added here too.

Spring Remoting and Web Services

```
@Service
@WebService
public class DefaultBidService implements BidService {
  @PersistenceContext
  private EntityManager entityManager;

  @Transactional
  public void addBid(Bid bid) {
    entityManager.persist(bid);
  }
}

public interface BidService {
  void addBid(Bid bid);
}
```

Spring Remoting and Web Services Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:ws="http://jax-ws.dev.java.net/spring/core"
    xmlns:wss="http://jax-ws.dev.java.net/spring/servlet"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
    http://jax-ws.dev.java.net/spring/core
    http://jax-ws.dev.java.net/spring/core.xsd
    http://jax-ws.dev.java.net/spring/servlet
    http://jax-ws.dev.java.net/spring/servlet.xsd">
  ...
  <wss:binding url="/bidService">
    <wss:service>
      <ws:service bean="#defaultBidService"/>
    </wss:service>
  </wss:binding>
  <bean class="org.springframework.remoting.rmi.RmiServiceExporter">
    <property name="serviceName" value="bidService"/>
    <property name="service" ref="defaultBidService"/>
    <property name="serviceInterface" value="com.actionbazaar.BidService"/>
  </bean>
</beans>
```

Configuration via JAX-WS RI schema

More traditional Spring exporter for remoting

Spring Remoting and Web Services web.xml Configuration

```
<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      classpath*:**/applicationContext*.xml
    </param-value>
  </context-param>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>jaxws-servlet</servlet-name>
    <servlet-class>
      com.sun.xml.ws.transport.http.servlet.WSSpringServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>jaxws-servlet</servlet-name>
    <url-pattern>/bidService</url-pattern>
  </servlet-mapping>
</web-app>
```

The Spring container getting bootstrapped.

Need to add a mapping for each web service

Java EE Scheduling

```
@Stateless
public class NewsLetterGenerator {
   @Schedule(dayOfMonth="Last Tues",
     month="Jan-May, Sep-Nov",
     timezone="America/New_York")
   public void generateMonthlyNewsLetter() {
      ...
   }
}
```

No further configuration needed

Spring Scheduling

```
@Component
public class NewsLetterGenerator {
  ...
  public void generateMonthlyNewsLetter() {
    ...
  }
}

<beans...>
  ...
  <bean id="jobDetail"
      class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
    <property name="targetObject" ref="newsLetterGenerator"/>
    <property name="targetMethod" value="generateMonthlyNewsLetter"/>
  </bean>
  <bean id="cronTrigger"
      class="org.springframework.scheduling.quartz.CronTriggerBean">
    <property name="jobDetail" ref="jobDetail"/>
    <property name="cronExpression" value="0 0 0 ? JAN-MAY,SEP-NOV 3L" />
    <property name="timeZone">
      <bean class="java.util.TimeZone" factory-method="getTimeZone">
        <constructor-arg value="America/New_York"/>
      </bean>
    </property>
  </bean>
  <bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
    <property name="triggers">
      <list><ref bean="cronTrigger"/></list>
    </property>
  </bean>
</beans>
```

More fine-grained control but with greater complexity.

Java EE Messaging

```
@MessageDriven(activationConfig={@ActivationConfigProperty(
    propertyName="destination",
    propertyValue="jms/OrderQueue")})
public class OrderProcessor implements MessageListener {
  @Current private OrderService orderService;

  public void onMessage(Message message) {
    try {
      ObjectMessage objectMessage = (ObjectMessage) message;
      Order order = (Order) objectMessage.getObject();
      orderService.addOrder(order);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

No further configuration needed

Spring Messaging

```
@Component
public class OrderProcessor implements MessageListener {
  @Autowired private OrderService orderService;

  public void onMessage(Message message) {
    try {
      ObjectMessage objectMessage = (ObjectMessage) message;
      Order order = (Order) objectMessage.getObject();
      orderService.addOrder(order);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Spring Messaging Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jms="http://www.springframework.org/schema/jms"
    xmlns:amq="http://activemq.apache.org/schema/core"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd
    http://www.springframework.org/schema/aop
    http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
    http://www.springframework.org/schema/jms
    http://www.springframework.org/schema/jms/spring-jms-2.5.xsd
    http://activemq.apache.org/schema/core
    http://activemq.apache.org/schema/core/activemq-core.xsd">
  ...
  <amq:broker useJmx="false" persistent="false">
    <amq:transportConnectors>
      <amq:transportConnector uri="tcp://localhost:0" />
    </amq:transportConnectors>
  </amq:broker>
  <amq:queue id="jms/OrderQueue" physicalName="queue.OrderQueue"/>
  <amq:connectionFactory id="connectionFactory" brokerURL="vm://localhost"/>
  <jms:listener-container acknowledge="transacted" concurrency="3-5">
    <jms:listener destination="jms/OrderQueue" ref="orderProcessor"/>
  </jms:listener-container>
</beans>
```

Outside a Java EE environment, a JMS is configured

A JTA transaction must be used for robust message handling (maybe JOTM is an option?)

Each message listener must be configured with the JMS container

Spring Message Producer

```
@Service
public class OrderService {
  private JmsTemplate jmsTemplate;
  private Queue queue;

  @Autowired
  public void setConnectionFactory(
      ConnectionFactory connectionFactory) {
    this.jmsTemplate = new JmsTemplate(connectionFactory);
  }

  @Resource(name=”jms/OrderQueue”)
  public void setQueue(Queue queue) {
    this.queue = queue;
  }

  @Transactional
  public void sendOrder(Order order) {
    this.jmsTemplate.send(queue, new MessageCreator() {
      public Message createMessage(Session session)
          throws JMSException {
        return session.createObjectMessage(order);
      }
    });
  }
}
```

Note, this is not coordinated with any other resources unless using JTA

Note, session and connection is opened/closed per call, so connection pooling is critical

Java EE Message Producer

```
@Stateless
public class OrderService {
  @OrderSession private Session session;
  @OrderMessageProducer private MessageProducer producer;

  public void sendOrder(Order order) {
    try {
      ObjectMessage message = session.createObjectMessage();
      message.setObject(order);
      producer.send(message);
    } catch (JMSException e) {
      e.printStackTrace();
    }
  }
}
```

DI based JMS abstractions

Java EE JMS Abstraction

```
public class OrderResources {
  @Resource(name="jms/ConnectionFactory")
  private ConnectionFactory connectionFactory;

  @Resource(name="jms/OrderQueue")
  private Queue orderQueue;

  @Produces @OrderConnection
  public Connection createOrderConnection() throws JMSException {
    return connectionFactory.createConnection();
  }

  public void closeOrderConnection(
      @Disposes @OrderConnection Connection connection)
          throws JMSException {
    connection.close();
  }

  @Produces @OrderSession
  public Session createOrderSession(
      @OrderConnection Connection connection) throws JMSException {
    return connection.createSession(true, Session.AUTO_ACKNOWLEDGE);
  }

  public void closeOrderSession(
      @Disposes @OrderSession Session session) throws JMSException {
    session.close();
  }

  @Produces @OrderMessageProducer
  public MessageProducer createOrderMessageProducer(
      @OrderSession Session session) throws JMSException {
    return session.createProducer(orderQueue);
  }

  public void closeOrderMessageProducer(
      @Disposes @OrderMessageProducer MessageProducer producer)
          throws JMSException {
    producer.close();
  }
}
```

The Birds-Eye View Again

- Java EE and Spring are both capable middleware solutions
- Functionally equivalent for the most part, with very different approaches
- For very vanilla applications, code is almost identical

Java EE

- Fully integrated platform with strong intelligent defaulting and minimal configuration (convention-over-configuration)
- Non-redundant APIs with specialized roles
- Focus on annotations for meta-data and type-safety
- Widely supported open standard
- Vendor and platform agnostic code
- Integration with non-standard technologies left to vendors
- Vendors free to innovate via non-standard extensions and value-added features like clustering, HA, administration, monitoring, development/deployment tools

Spring Framework

- Explicit configuration, typically through XML
- More flexibility for fine-grained control but more complexity
- Ease-of-use abstractions over lower-level, general purpose APIs like JDBC, JMS and JavaMail
- Non-standard open source technology from a commercial vendor
- Integration with standard as well as non-standard technologies that are often overlapping
- Portability across runtime platforms at the risk of vendor lock-in

- There is no practical reason to be forced to choose between the two
- Balanced competition in server-side Java is good for both developers and vendors
- There are excellent integration possibilities

Summary

- Both are competent middleware stacks with mostly equivalent functionality but very different approaches
- Java EE is a tightly integrated platform with intelligent defaulting, minimal configuration and specialized, non-overlapping APIs.
- Spring is a pluggable framework with explicit configuration, fine grained control and wide-variety of integration with overlapping technologies
- Java EE is annotation centric while Spring is XML-bound
- One is an open standard, while the other is an open source tool from a commercial vendor

QUESTIONS?

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