Spring in Action

Craig Walls

Spring Dallas User Group

August 15, 2007

craig-sia@habuma.com

These slides: http://www.habuma.com/spring/sia-sdug.pdf

About you…

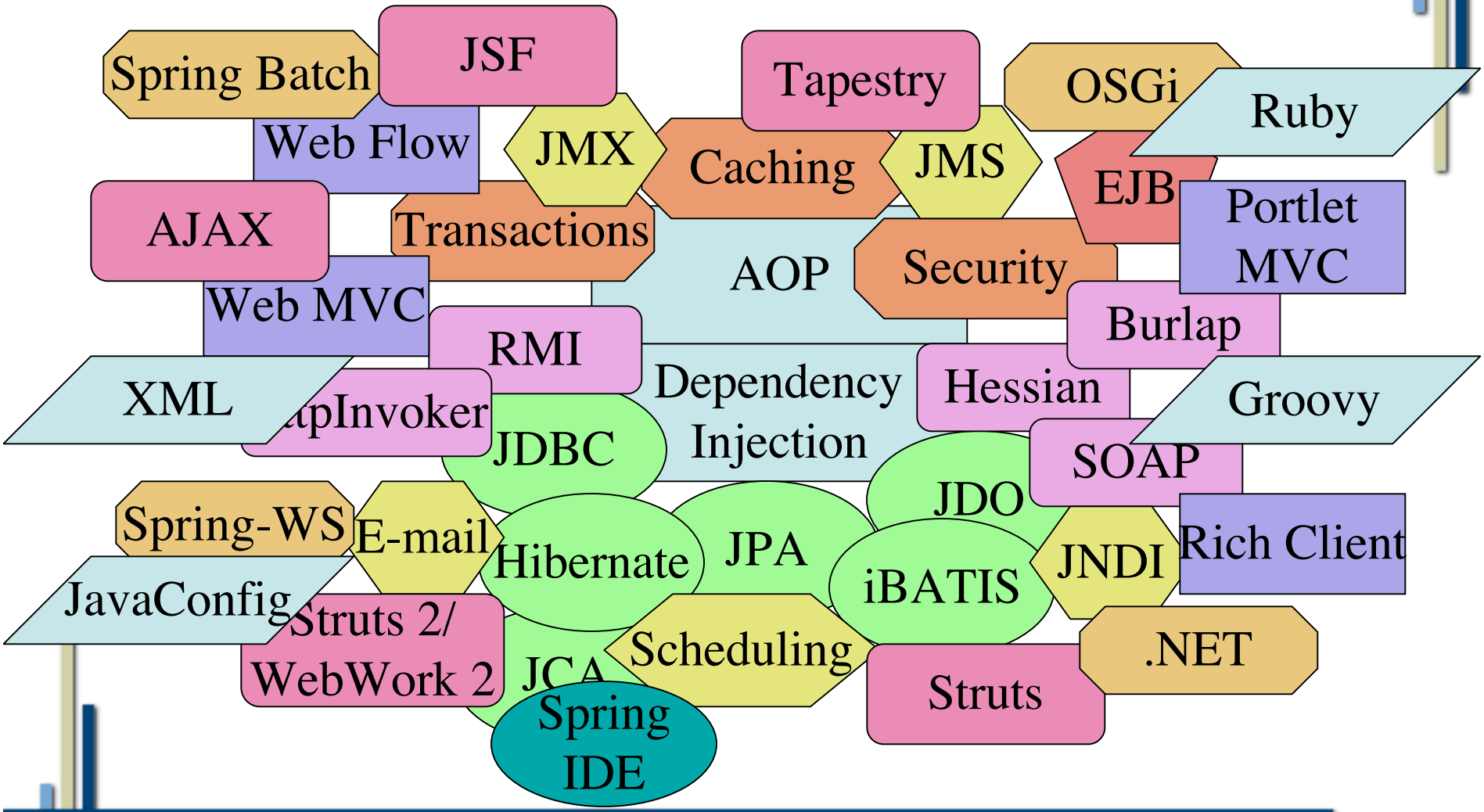
- Java? .NET? Ruby/Rails?
  - Java 6? Java 5? Java 1.4? Java 1.3? 1.2 or older?
- Who's using Spring? How long?
  - Spring 1.2? Spring 2? Spring 2.1?

What is Spring?

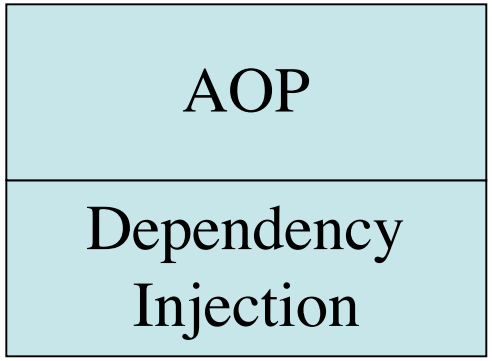
Spring is…

- A POJO container?
- A lightweight framework?
- A platform?
- A catalyst for change?

Spring does…


Spring Batch
JSF
Tapestry
OSGi
Ruby
Web Flow
JMX
Caching
JMS
EJB
AJAX
Transactions
AOP
Security
Portlet MVC
Web MVC
RMI
Burlap
XML
HttpInvoker
Dependency Injection
Hessian
Groovy
JDBC
SOAP
JDO
Spring-WS
E-mail
Hibernate
JPA
iBATIS
JNDI
Rich Client
JavaConfig
Struts 2/ WebWork 2
JCA
Scheduling
.NET
Struts
Spring IDE

Spring does…



What's new in Spring 2

- New/extensible configuration namespaces
- Easier AOP and support for @AspectJ
- Easier transactions
- Support for JPA
- Asynchronous JMS support
- New JDBC templates (Java 5 and named parameters)
- New form-binding JSP tag library
- Portlet MVC
- Dynamic language beans (JRuby, Groovy, BeanShell)
- JMX: Notifications and registration control
- Convention over configuration improvements

What's coming in Spring 2.1

- JMS configuration namespace
- Context configuration namespace
  - <aop:spring-configured> moves to <context:spring-configured>
- Annotation-driven configuration
  - @Component, @Autowired
  - JSR-250: @Resource, @PostConstructor, @PreDestroy
- Autodetected components
- Requires JDK 1.4 or higher
- Named parameters added to SimpleJdbcTemplate
- ParameterizedBeanPropertyRowMapper for automatically mapping between columns and bean properties
- <context:load-time-weaver/> : Spring configured load time weaving
- Hibernate 2.1 support goes away
- JUnit 4.x support (in next milestone)
- Spring 2.5 ???

Dependency Injection

DI in a nutshell

- Without DI, objects get their own dependencies
  - Directly through construction
  - Through factories
- With DI, objects are *given* their dependencies
  - Through constructors
  - Through setter methods
  - Indirectly through method replacement
- DI + interfaces = loosely coupled objects

Knights and Quests

- Imagine a ValiantKnight that embarks on quests
- A ValiantKnight needs a Quest
- Quest is an interface:

```
public interface Quest {
  void embark();
}
```

What's wrong with this?

```
public class ValiantKnight {
  public void embarkOnQuest() {
    Quest quest = new SlayDragonQuest();
    quest.embark();
  }
}
```

How about this?

```
public class ValiantKnight{
  public void embarkOnQuest() {
    Quest quest = QuestFactory.
            getInstance().getQuest();
    quest.embark();
  }
}
```

Is this any better?

```
public class ValiantKnight {
  public void embarkOnQuest() {
    InitialContext ctx = null;
    try {
      ctx = new InitialContext();
      Quest quest = (Quest) ctx.lookup(
          "java:comp/env/Quest");
      quest.embark();
    } catch (NamingException e) {
    } finally {
      if(ctx != null) {
        try {ctx.close(); }
        catch (Exception e) {}
      }
    }
  }
}
```

Let's try this…

```
public class ValiantKnight {
  private Quest quest;

  public ValiantKnight(Quest quest) {
    this.quest = quest;
  }

  public void embarkOnQuest() {
    quest.embark();
  }
}
```

Or perhaps…

```
public class ValiantKnight {
  private Quest quest;

  public ValiantKnight() {}

  public void setQuest(Quest quest) {
    this.quest = quest;
  }

  public void embarkOnQuest() {
    quest.embark();
  }
}
```

Where does Quest come from?

How is it Implemented?

Wiring in Spring (constructor)

```
<bean id="knight"
      class="com.springinaction.ValiantKnight">
  <constructor-arg ref="quest" />
</bean>

<bean id="quest"
      class="com.springinaction.SlayDragonQuest" />
```

Wiring in Spring (setter)

```
<bean id="knight"
      class="com.springinaction.ValiantKnight">
    <property name="quest" ref="quest" />
</bean>

<bean id="quest"
      class="com.springinaction.SlayDragonQuest" />
```

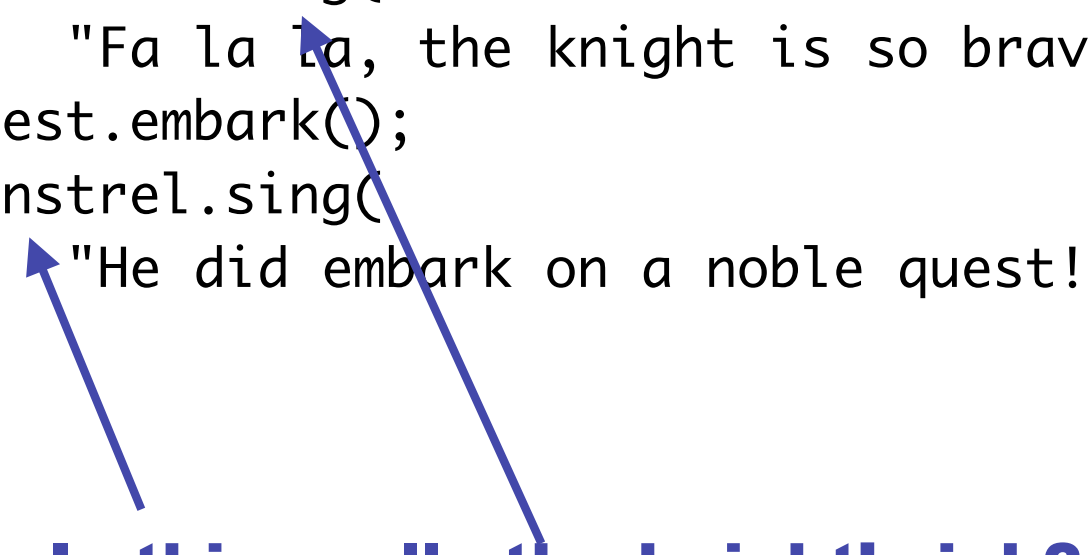
Aspect-Oriented Programming

AOP in a nutshell

- Aspects decouple “concerns” from the objects that they apply to
- Common examples: Logging, caching, security, transactions
- Imagine a Minstrel class that chronicles a Knight’s exploits in song…

Without AOP

```
public void embarkOnQuest() {
  minstrel.sing(
      "Fa la la, the knight is so brave!");
  quest.embark();
  minstrel.sing(
      "He did embark on a noble quest!");
}
```



With AOP

```
public void embarkOnQuest() {
  quest.embark();
}
```

Where's the Minstrel?

Minstrel.java

```
public class Minstrel {
  public void singBefore() {
    System.out.println(
        "Fa la la, the knight is so brave!");
  }

  public void singAfter() {
    System.out.println(
        "He did embark on a noble quest!");
  }
}
```

Weaving aspects in Spring

```
<bean id="minstrel" class="com.springinaction.Minstrel" />

<aop:config>
  <aop:aspect ref="minstrel">
    <aop:pointcut
        id="embarkment"
        expression="execution(* *.embarkOnQuest(..))" />
    <aop:before
        method="singBefore"
        pointcut-ref="embarkment" />
    <aop:after-returning
        method="singAfter"
        pointcut-ref="embarkment" />
  </aop:aspect>
</aop:config>
```

Using @AspectJ

```
@Aspect
public class Minstrel {
  @Pointcut("execution(* *.embarkOnQuest(..))")
  public void embarkment() {}

  @Before("embarkment()")
  public void singBefore() {
    System.out.println(
        "Fa la la, the knight is so brave!");
  }

  @After("embarkment()")
  public void singAfter() {
    System.out.println(
        "He did embark on a noble quest!");
  }
}
```

In Spring XML…

```
<aop:aspectj-autoproxy />
```

Yep…that's it

DI meets AOP

- @Configurable enables injection into objects not managed by Spring
  - Domain objects, for example
- Configure in Spring XML:
  - Spring 2.0: `<aop:spring-configured/>`
  - Spring 2.1: `<context:spring-configured/>`
- Needs a load-time-weaver:
  - Spring 2.0: -javaagent:/path/to/aspect-weaver.jar
  - Spring 2.1…two options:
    - -javaagent:/path/to/spring-agent.jar
    - `<context:load-time-weaver />`

Spring and JDBC

Conventional JDBC

```
Connection conn = null;
PreparedStatement stmt = null;
ResultSet rs = null;
try {
  conn = dataSource.getConnection();
  stmt = conn.prepareStatement("select id, first_name, last_name from Employee where id=?");
  stmt.setInt(1, id);
  rs.stmt.executeQuery();
  Employee employee = null;
  if(rs.next()) {
    employee = new Employee();
    employee.setId(rs.getInt(1));
    employee.setFirstName(rs.getString(2));
    employee.setLastName(rs.getString(3));
  }
  return employee;
} catch (SQLException e) {

} finally {
  try {
    if(rs ! null) { rs.close(); }
    if(stmt != null) { stmt.close(); }
    if(conn != null) { conn.close(); }
  } catch (SQLException e) {}
}
```

Déjà vu?

What do you intend to do about this???

SQLException

- The case of the ambiguous and useless checked exception…
  - SQLException means that something went wrong with the database…but what?
  - The types of problems that SQLException represent are usually not runtime-addressable.
    - What can you do in a catch block to handle a column not found error?

Spring's DataAccessException

- CannotAcquireLockException
- CannotSerializeTransactionException
- CleanupFailureDataAccessException
- ConcurrencyFailureException
- DataAccessException
- DataAccessResourceFailureException
- DataIntegrityViolationException
- DataRetrievalFailureException
- DeadlockLoserDataAccesException
- EmptyResultDataAccessException
- IncorrectResultSizeDataAccessException
- IncorrectUpdateSemanticsDataAccessException
- InvalidDataAccessApiUsageException
- InvalidDataAccessResourceUsageException
- OptimisticLockingFailureException
- PermissionDeniedDataAccessException
- PessimisticLockingFailureException
- TypeMismatchDataAccessException
- UncategorizedDataAccessException

JDBC: Spring-style

```
List matches = jdbcTemplate.query(
    "select id, first_name, last_name from Employee" +
    " where id=?",
    new Object[] {Long.valueOf(id)},
    new RowMapper() {
      public Object mapRow(ResultSet rs, int rowNum)
          throws SQLException, DataAccessException {
        Employee employee = new Employee();
        employee.setId(rs.getInt(1));
        employee.setFirstName(rs.getString(2));
        employee.setLastName(rs.getString(3));
        return employee;
      }
    });

return matches.size() > 0 ?
    (Employee) matches.get(0) : null;
```

Notice no awkward try/catch block

And no JDBC boilerplate

JDBC Template

```
<bean id="dataSource"
  class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
  <property name="driverClassName”
     value="org.hsqldb.jdbcDriver" />
  <property name="url"
     value="jdbc:hsqldb:hsql://localhost/employee/employee" />
  <property name="username" value="sa" />
  <property name="password" value="" />
</bean>

<bean id="jdbcTemplate"
    class="org.springframework.jdbc.core.JdbcTemplate" >
  <property name="dataSource" ref="dataSource" />
</bean>

<bean id="employeeDao" class="com.springinaction.JdbcEmployeeDao">
  <property name="jdbcTemplate" ref="jdbcTemplate" />
</bean>
```

JDBC: Spring 2 and Java 5

```
List<Employee> matches = simpleJdbcTemplate.query(
    "select id, first_name, last_name from Employee" +
        " where id=?",
    new ParameterizedRowMapper<Employee>() {
      public Employee mapRow(ResultSet rs, int rowNum)
          throws SQLException {
        Employee employee = new Employee();
        employee.setId(rs.getInt(1));
        employee.setFirstName(rs.getString(2));
        employee.setLastName(rs.getString(3));
        return employee;
      }
    },
    id);

return matches.size() > 0 ? matches.get(0) : null;
```

Uses generics

No need to wrap parameters

No need to cast result

One more thing on templates

- JDBC isn't the only place where Spring provides templates…
  - Hibernate
  - JPA
  - JDO
  - TopLink
  - iBATIS
  - JCA
  - JMS
  - JNDI

Declarative Transactions

EJB's killer feature…

- …now available in convenient POJO form.
- Spring uses AOP to enable declarative transactions on POJO methods
  - In some ways, even more capable than EJB transactions (more propagation options, isolation levels…)

Spring 2 declarative tx

```
<tx:advice id="txAdvice">
  <tx:attributes>
    <tx:method name="add*" propagation="REQUIRED" />
    <tx:method name="*" propagation="SUPPORTS"
        read-only="true"/>
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:advisor
      advice-ref="txAdvice"
      pointcut="execution(* *..EmployeeService.*(..))" />
</aop:config>
```

Spring 2 + Java 5 transactions

In service class:

```
@Transactional(propagation=Propagation.SUPPORTS, readOnly=true)
public class EmployeeServiceImpl implements EmployeeService {
…
   @Transactional(propagation=Propagation.REQUIRED,
    readOnly=false)
   public void addEmployee(Employee rant) {
   …
   }
…
}
```

In Spring XML:

```
<tx:annotation-driven />
```

Spring MVC

Why Spring MVC?

- Spring usually doesn't reinvent the wheel…
  - No persistence framework
- At one time, Struts was only option
  - And not necessarily a good option
- Spring MVC reinvented the MVC wheel…addressing the shortcomings of Struts

Struts 1.x Action

```
public class DisplayCustomerAction extends Action {
   public ActionForward execute(ActionMapping mapping,
       ActionForm form, HttpServletRequest request,
       HttpServletResponse response) throws Exception {

      CustomerDetailForm cdf =
              (CustomerDetailForm) form;
      Customer customer =
              lookupCustomer(cdf.getAccountNumber());
      request.setAttribute("customer", customer);

      return mapping.findForward("customerDetail");
   }
}
```

Spring MVC Controller

```
public class DisplayCustomerController
    extends AbstractController {

  protected ModelAndView handleRequestInternal(
      HttpServletRequest req,
      HttpServletResponse response) throws Exception {

      long customerId =
          Long.valueOf(req.getParameter("id"));
      Customer customer = lookupCustomer(customerId);

      return new ModelAndView(
          "customerDetail", customer);
    }
}
```

Mapping URLs to controllers

- SimpleUrlHandlerMapping:

```
<bean id="urlMapping"
      class="org.springframework.web.servlet.handler.
                                    SimpleUrlHandlerMapping">
  <property name="mappings">
    <value>
      /home.htm=homeController
      /login.htm=loginController
      /addSpittle.htm=addSpittleController
      /addSpitter.htm=addSpitterController
    </value>
  </property>
</bean>
```

Auto-mapping of controllers

- ControllerClassNameHandlerMapping:

```
<bean id="urlMapping"
    class="org.springframework.web.servlet.mvc.support.
                    ControllerClassNameHandlerMapping" />
```

  - /home.htm → HomeController
  - /login.htm → LoginController
  - /addspittle.htm → AddSpittleController

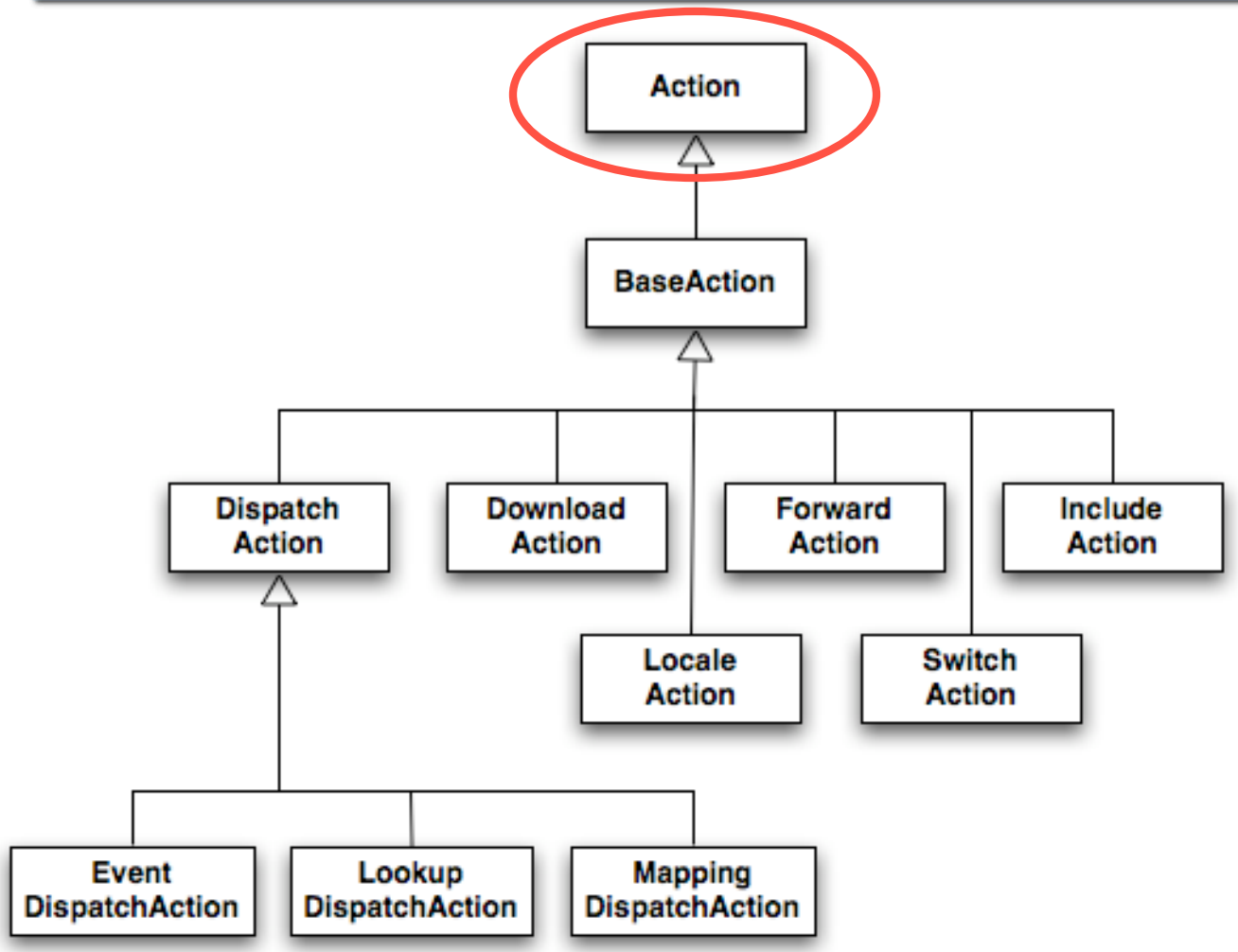
Mapping view names to views

- InternalResourceViewResolver:

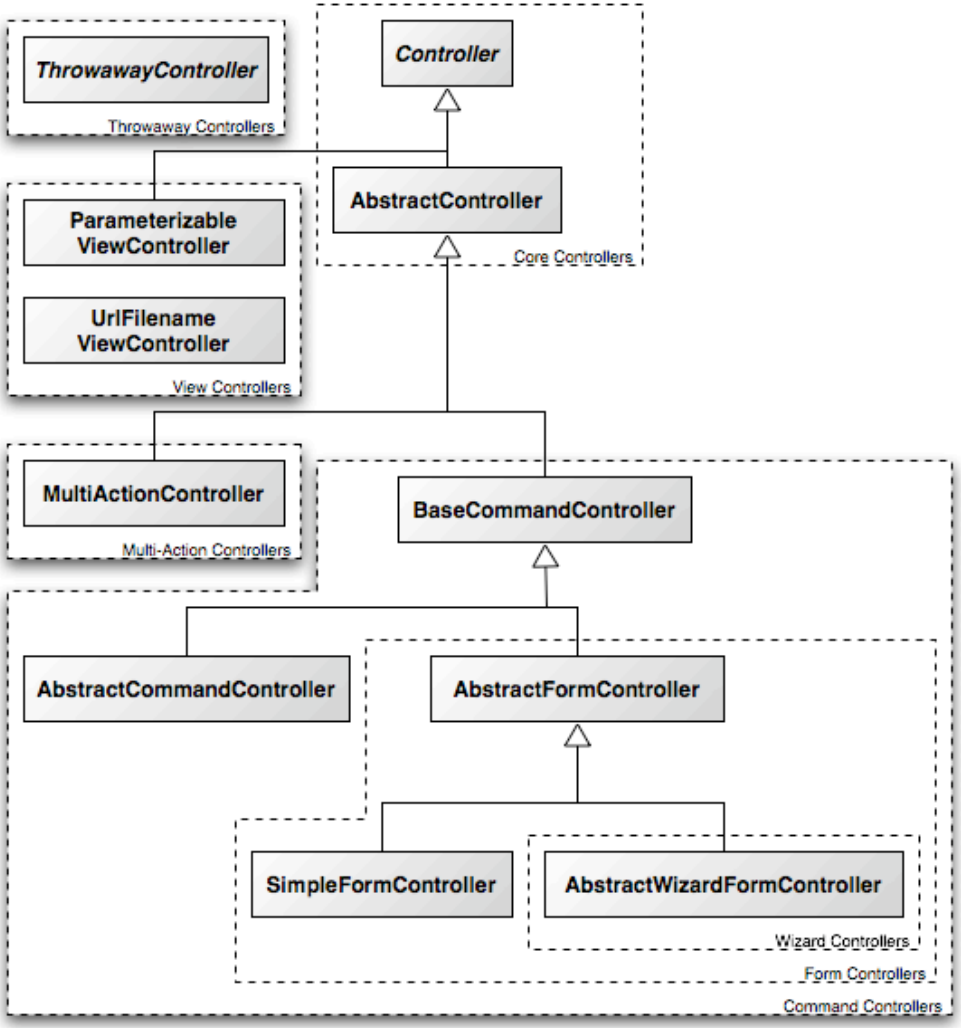
```
<bean id="jspViewResolver"
    class="org.springframework.web.servlet.view.
                                        InternalResourceViewResolver">
  <property name="prefix" value="/WEB-INF/jsp/" />
  <property name="suffix" value=".jsp" />
</bean>
```

  - "home" → /WEB-INF/jsp/home.jsp
  - "login" → /WEB-INF/jsp/login.jsp
  - "orderForm" → /WEB-INF/jsp/orderForm.jsp

The Struts 1.x Action selection



Spring Controller Selection



What about…???

- Struts 2/WebWork 2?
- JSF?
- Tapestry?
- Wicket?
- Seam?
- Grails?
- The billions of other MVC frameworks

Q & A

http://www.springinaction.com

craig-sia@habuma.com