Welcome

Overview of the Spring Framework

Ernest Hill
ernesthill@earthlink.net

Why the Spring Framework

- Design Techniques and Coding Standards for J2EE Projects
  - http://www.theserverside.com/articles/content/RodJohnsonInterview/JohnsonChapter4.pdf
- Introducing the SpringFramework
  - http://www.theserverside.com/resources/article.jsp?l=SpringFramework

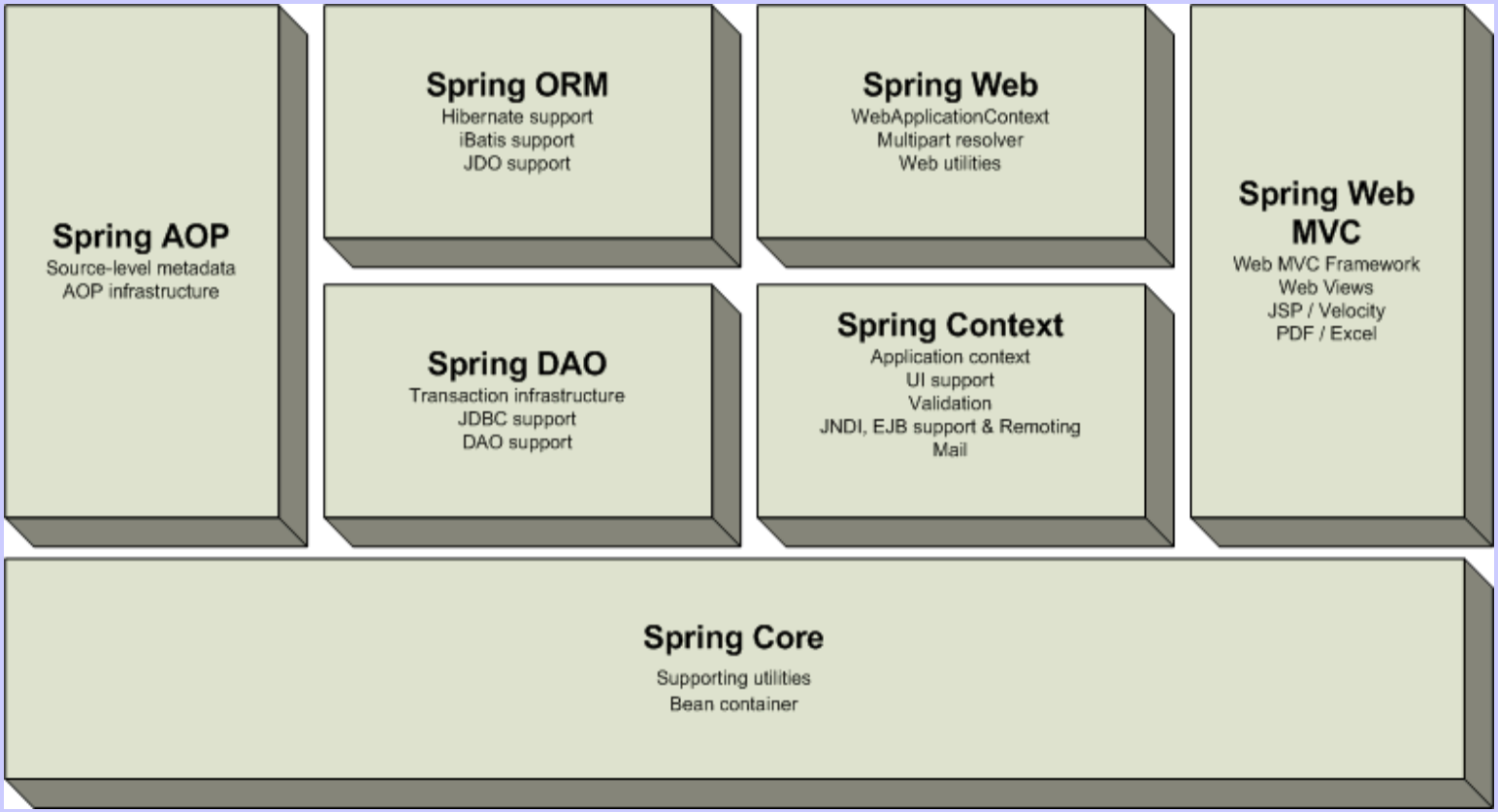
What is the Spring Framework?

- A open source project whose main goals are to:
  1) Make J2EE easier to use by providing a layered architecture that is internally consistent
  2) Promote good programming practices.

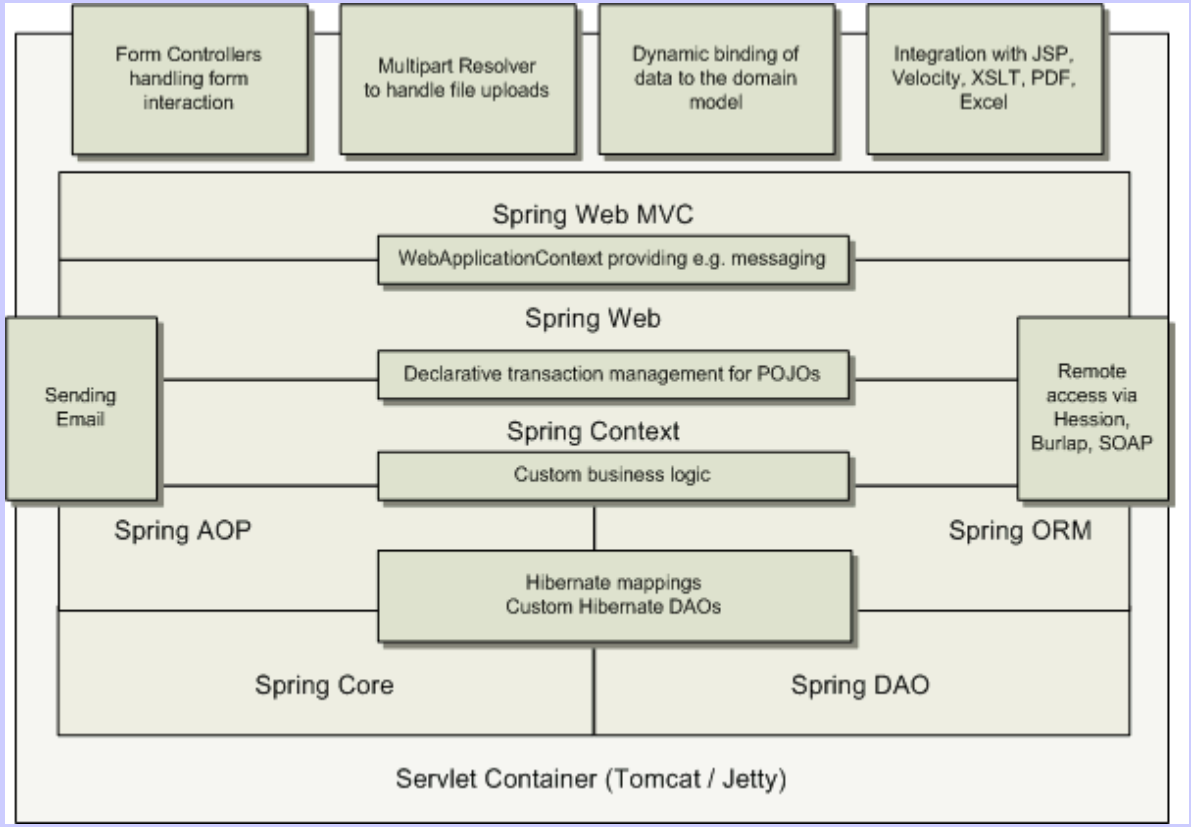
How does it achieve it's goal?

- By providing a set of modules that can be used in concert or individually

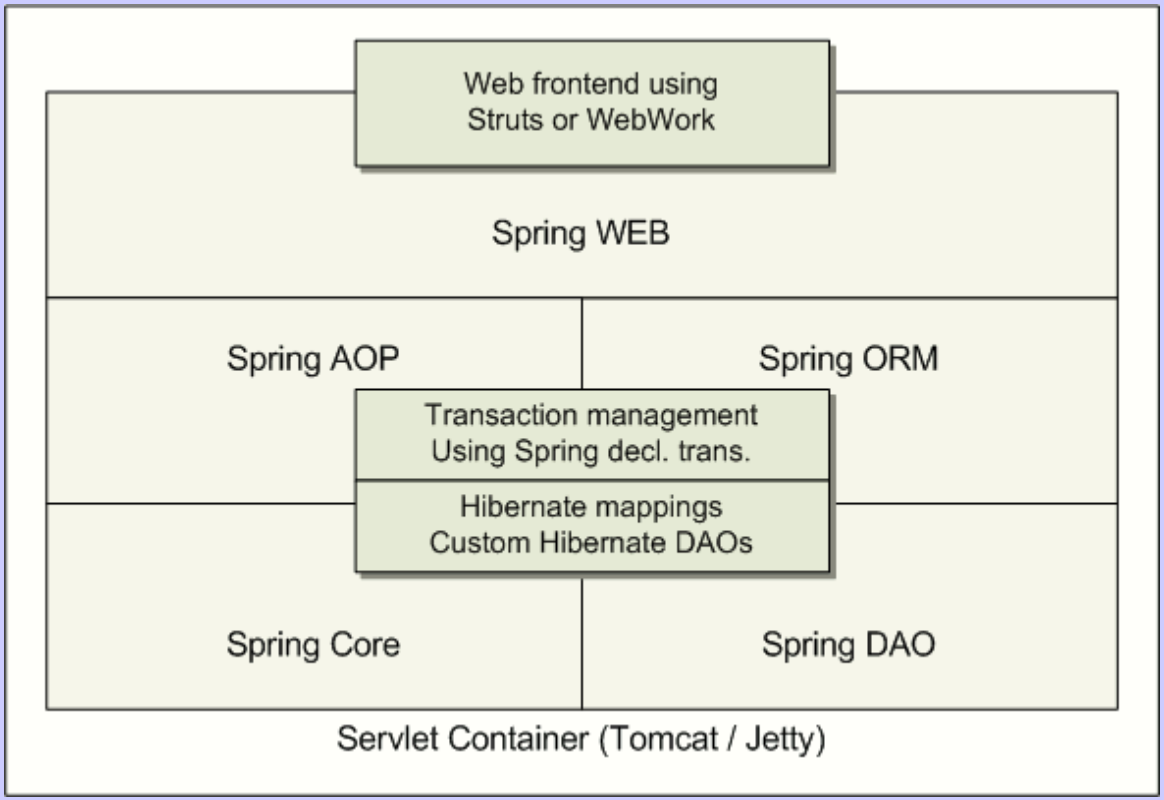
Spring Overview



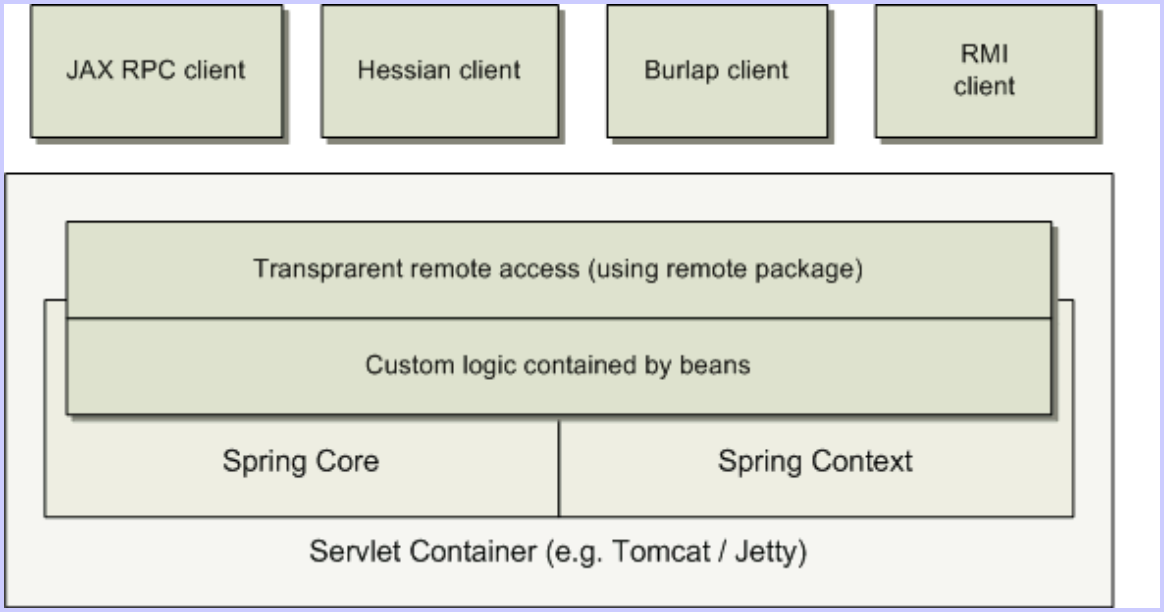
Full-Fledged Spring Web Application Usage



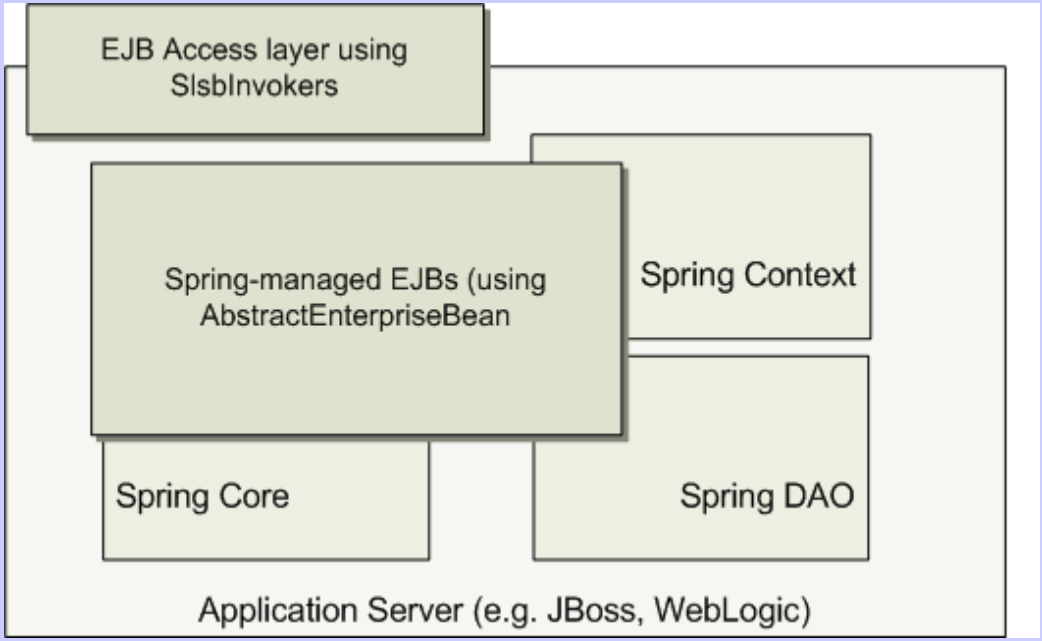
Middle Tier


Web frontend using
Struts or WebWork
Spring WEB
Spring AOP
Spring ORM
Transaction management
Using Spring decl. trans.
Hibernate mappings
Custom Hibernate DAOs
Spring Core
Spring DAO
Servlet Container (Tomcat / Jetty)

Transparent Remote Access



EJB Abstraction Layer



Spring Core

- Designed for working with JavaBeans
- Via BeanFactory

Bean Factory

- Light weight container
- Loads Bean Definitions
- Enables object to be retrieved by name
- Instantiates one or more instance of a bean
- Manages relationships between objects

Bean Definition

- Defines the bean class
- The bean identifiers
- To singleton or not
- Properties and collaborators
- Constructor arguments
- Initialization method
- Destruction method

Inversion of Control/ Dependency Injection

- Beans do not do a lookup to resolve dependencies
- The container injects the dependency
- Two major variants
  - Setter-based dependency injection
  - Constructor-based dependency injection

Setter-based dependency injection

- Container calls setters on the bean after invoking an argumentless constructor
- Beans based on setter-based dependency are true JavaBeans
- Use of setter-based dependency is advocated by Spring

Example of Setter Based Injection

```
<bean id=”exampleBean” class=”examples.ExampleBean”>
   <property name=”beanOne”><ref bean=”anotherBean”/></property>
   <propetry name=”beanTwo”><ref bean=”someBean” /></propery>
   <property name="integerProperty">1</property>
</bean>

<bean id=”anotherBean” class=”examples.AnotherBean” />
<bean id=”someBean” class=”examples.someBean” />
```

```
public class ExampleBean {
    private AnotherBean beanOne;
    private someBean beanTwo;
    private int i;

    public void setBeanOne(AnotherBean beanOne) { this.beanOne = beanOne; }

    public void setBeanTwo(SomeBean beanTwo ) { this.beanTwo = beanTwo; }

    public void setIntegerProperty(int i ) { this.i = i; }
}
```

Constructor-based Dependency Injection

- Realized by invoking a constructor with a number of arguments
- May be prefered as means of insuring that beans can not be constructed in an invalid state

Example of Constructor Based Injection

```
<bean id=”exampleBean” class=”examples.ExampleBean”>
    <constructor-arg><ref bean=”anotherBean”/></constructor-arg>
    <constructor-arg><ref bean=”someBean”/></constructor-arg>
    <constructor-arg>1</constructor-arg>
</bean>

<bean id=”anotherBean” class=”examples.AnotherBean” />
<bean id=”someBean” class =”examples.SomeBean”/>


Public class ExampleBean {
    private AnotherBean beanOne;
    private SomeBean beanTwo;
    private int i;

    public ExampleBean ( AnotherBean anotherBean, SomeBean someBean,
      int i ) {
      this.beanOne = anotherBean;
      this.beanTwo = someBean;
      this.i = i;
    }
}
```

Advantages of Dependency Injection

- Components are simpler to write
- Components are easier to test
- Objects do not depend on the container API
- Objects can run outside the container

Example of BeanFactory Usage

```
InputStream inputStream = new FileInputStream(“beans.xml”);
BeanFactory factory = new XMLBeanFactory(inputStream);

// Check if a bean definition exists by the name exampleBean
if ( factory.containsBean(“exampleBean”) {
  // exampleBean definition exists
}

// Can throw NoSuchBeanDefinition
ExampleBean eb = (ExampleBean) factory.getBean(“exampleBean”);

// Can throw BeanNotOfRequiredTypeException
ExampleBean eb =
  (ExampleBean) factory.getBean(“exampleBean”, ExampleBean.class );

// Can throw NoSuchBeanDefinition
if ( factory.isSingleton(“exampleBean” ) {
  // this bean is a singleton
}

String[] aliasas = factory.getAliases(“exampleBean”);
```

Property Editors

- Convenient to represent a property in a human readable form and convert it back to object
- Property Editors must implement java.beans.PropertyEditor

Built-in Property Editors

- Standard Java Property Editors
  - Bool, Byte, Color, Double, Float, Font, Int, Long, Short, String
- Standard Spring
  - Class, File, Local, Properties, StringArray, URL
- Custom Spring
  - CustomBoolean, CustomDate, CustomNumber, StringTrimmer

Application Context

- BeanFactory functionality in a more frame-work-style
- can be created declaratively using for ex-ample a ContextLoader

Added functionality of the ApplicationContext

- MessageSource, providing access to messages in, i18n-style
- Access to resources, like URLs and files
- Event propagation to beans implementing the ApplicationListener interface
- Loading of multiple contexts, allowing some of them to be focused and used for example only by the web-layer of an application

Using the MessageSource

- ApplicationContext interface extends an interface called MessageSource
- When an ApplicationContext gets loaded, it automatically searches for a MessageSource bean defined in the context named messageSource
- If it can't find any source for messages, an empty StaticMessageSource will be instantiated

Message Source Interface

- String getMessage (String code, Object[] args, String default, Locale loc)
- String getMessage (String code, Object[] args, Locale loc)
- String getMessage(MessageSourceResolv-able resolvable, Locale locale)

Message Source Implementations

- ResourceBundleMessageSource
- ReloadableBundleMessageSource
- StaticMessageSource
- All implement HierachicalMessageSource

ResourceBundleMessageSource Example

```
<beans>
    <bean id="messageSource"

    class="org.springframework.context.support.ResourceBundleMessageS
    ource">
        <property name="basenames">
            <list>
                <value>format</value>
                <value>exceptions</value>
                <value>windows</value>
            </list>
        </property>
    </bean>
</beans>
```

Propagating Events

- Provided via the ApplicationEvent class and ApplicationListener interface
- Standard Observer Pattern

Built-in Events

- ContextRefreshedEvent
- ContextClosedEvent
- RequestHandledEvent

ContextRefreshedEvent

- Event published when the ApplicationContext is initialized or refreshed. Initialized here means that all beans are loaded, singletons are pre-instantiated and the ApplicationContext is ready for use

ContextClosedEvent

- Event published when the ApplicationCon-text is closed, using the close() method on the ApplicationContext. Closed here means that singletons are destroyed

RequestHandledEvent

- A web-specific event telling all beans that a HTTP request has been serviced (i.e. this will be published after the request has been finished). Note that this event is only applicable for web applications using Spring's DispatcherServlet

Custom Events

- All you need to do is call the publishEvent() method on the ApplicationContext

Custom Event Example Application Context

```
<bean id="emailer" class="example.EmailBean">
    <property name="blackList">
        <list>
            <value>black@list.org</value>
            <value>white@list.org</value>
            <value>john@doe.org</value>
        </list>
    </property>
</bean>

<bean id="blackListListener" class="example.BlackListNotifier">
    <property name="notificationAddress">
        <value>spam@list.org</value>
    </property>
</bean>
```

Custom Event Example
Actual Beans

```
public class EmailBean implements ApplicationContextAware {
   private List blackList;

   public void setBlackList(List blackList) { this.blackList = blackList; }

   public void setApplicationContext(ApplicationContext ctx) {this.ctx = ctx;}

   public void sendEmail(String address, String text) {
      if (blackList.contains(address)) {
         BlackListEvent evt = new BlackListEvent(address, text);
         ctx.publishEvent(evt);
         return;
      }
   }
}

public class BlackListNotifier implement ApplicationListener {
   private String notificationAddress;

   public void setNotificationAddress(String notificationAddress) {
      this.notificationAddress = notificationAddress;
   }
   public void onApplicationEvent(ApplicationEvent evt) {
      if (evt instanceof BlackListEvent) {// notify appropriate person }
   }
}
```

Using Resources Within Spring

- ApplicationContext implements Re-sourceLoader interface
- Resource getResource(String location)
  - Fully qualified URLs, “file:C:/test.data”
  - Relative File Paths, “WEB-INF/test.data”
  - Classpath psuedo URL “classpath:test.data”

Resource Interface

- exists() - Checks if the resource exists, returning false if it doesn't
- getInputStream() - Opens an InputStream on the resource and returns it
- IsOpen() - Return whether this resource represents a handle with an open stream. If true, the InputStream cannot be read multiple times, and must be read and closed to avoid resource leaks.
- GetDescription() - Returns a description of the resource, often the fully qualified file name or the actual URL
- GetFile() - Return a File handle for this resource

Extra marker interface

- public interface ApplicationContextAware
  - void setApplicationContext(ApplicationContext context)

Creating an ApplicationContext from a web application

- Can be created declaratively using for example a ContextLoader
- Can also be created programmatically

ContextLoader Implementations

- ContextLoaderListener
  - Servlet 2.4 and possible 2.3 compatible
- ContextLoaderServlet
  - Servlet 2.2 Compatible

ContextLoaderListener Example

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/daoContext.xml /WEB-
    INF/applicationContext.xml</param-value>
</context-param>

<listener>
    <listener-
    class>org.springframework.web.context.ContextLoaderListener</list
    ener-class>
</listener>
```

ContextLoaderServlet Example

```
<servlet>
  <servlet-name>context</servlet-name>
  <servlet-
     class>org.springframework.web.context.ContextLoaderServlet</se
     rvlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Spring AOP

- AOP decomposes programs into aspects

AOP Concepts

- Aspect
- Joinpoint
- Advice
- PointCut
- Introduction
- Target Object
- AOP proxy
- Weaving

Aspect

- The modularization a concern that might otherwise cut across multiple objects.
- Transaction management is a good exam-ple
- Spring implements Aspects as Advisors and Interceptors

JoinPoint

- A point during the execution a the program, such a when a method is invoked or an exception is thrown

Advice

- The action that is to be taken at a particular jointpoint.
- Types of advise include
  - Around
  - Before
  - Throws
  - After Returning
- Sprint maints a chain of 'interceptors' around the joinpoint

Pointcut

- A set of joinpoints specifying when an advice should execute

Introduction

- Adding methods of fields to an advised class
- Spring allows you to introduce new inter-faces to advised objects

Target

- Object containing the joinpoint
- Also referred to as the advised or proxied object

AOP Proxy

- The object created by the AOP framework that includes the advise.
- Spring will create a proxy using a JDK dynamic proxy or a CGLIB proxy.

Weaving

- Assembling the aspects to create an advised object.
- Can be done at compile time like AspectJ.
- Spring performs weaving at runtime

Spring AOP Capabilities

- Implemented in pure Java
- Suitable for use in J2EE container since does not need to control the class loader hierachy
- Spring supports interception of methods
- Does not support field interception
- Provides classes to represent pointcuts and different advise types
- Spring advises object at instance, rather than class loader level

Spring DAO

- Aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO
- Allows you to switch between them fairly easily

Consistent Exception Hierarchy

- Provides an API that moves tedious error handling out of the application into the framework
- Wraps exceptions converting them from proprietary checked exceptions to abstract-ed runtime exceptions
- Examines metadata to ascertain the database uses this knowledge to map ex-ception to correct exception in hierarchy

Consistent Abstract Classes for DAO Support

- JdbcDaoSupport
  - Requires a datasource to be set
- HibernateDaoSupport
  - Requires a SessionFactory to be set
- JdoSDaoSupport
  - Requires a PersistenceManager to be set

Spring Web MVC

- Powerful and highly configurable
- Designed around a DispatcherServlet
- Unlike Struts which forces your Action and Form object into concrete inheritance, Spring MVC is entirely based on interfaces
- View agnostic, you don't get pushed into JSP
- SpringControllers are configured via De-pendency Injection

Features of Spring Web MVC

- Clear separation of roles: controller vs validator vs command object vs form object vs model object
- Straightforward configuration of both framework and application classes as JavaBeans
- Non-intrusive, use whatever Controller you need (plain, command, form, wizard, multi-action or custom) instead of Action/ActionForm for everything

Features of Spring Web MVC

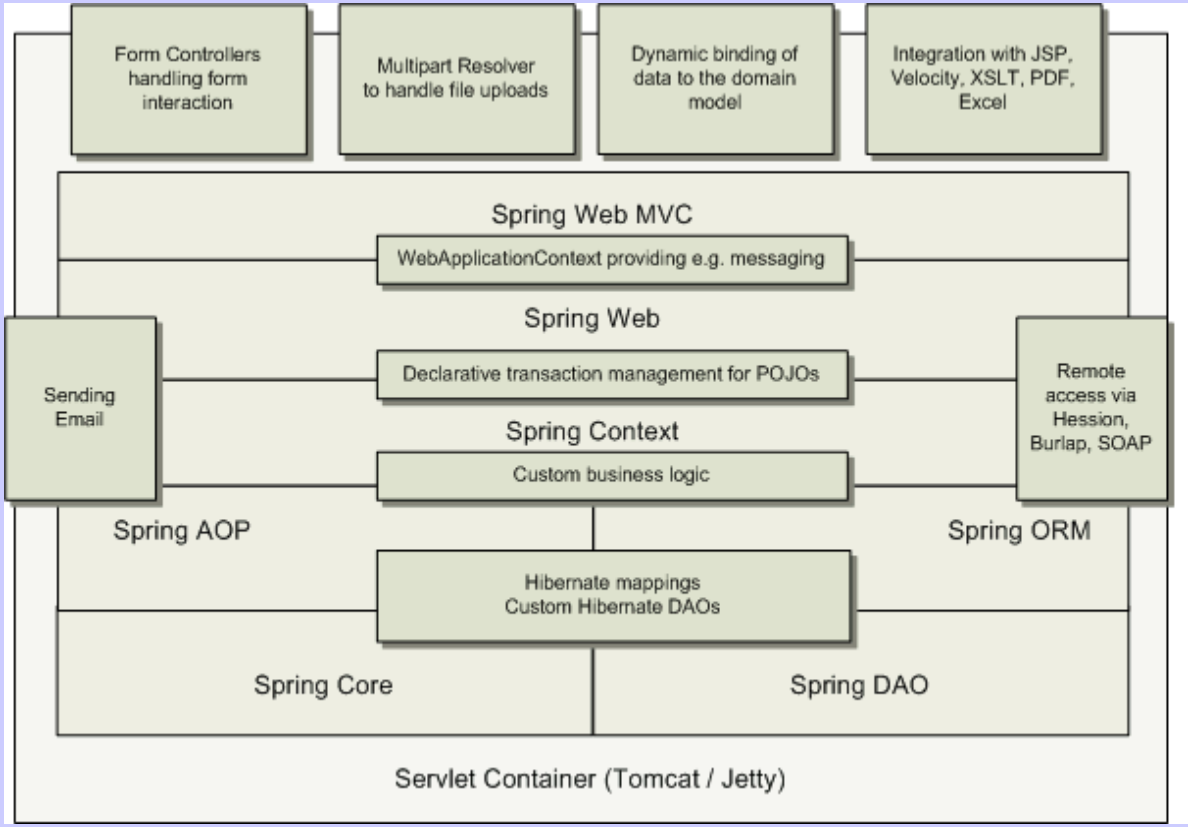
- Resuable business code, no need for duplication. Use existing business objects as commands or form objects instead of mirroring them in special ActionForm classes.
- Handler and view mapping strategies from simple to sophisticated instead of one way
- Related Projects
- Rich Client Platform
  - Sprint RCP
- Validation
  - Commons-validator
  - Attribute based

Resources

- The Spring web site
  - http://www.springframework.org
- Introducing the Spring Framework
  - http://www.theserverside.com/articles/article.tss?l=SpringFramework
- Expert one-on-one J2EE Design and Development
  - By Rod Johnson
  - Chapter 4 for available at http://www.theserverside.com/articles/content/Rod

Resources

- Expert One-on-one J2EE Development without EJB
  - By Rod Johnson and Jürgen Höller
- Developing a Spring Framework MVC application step-by-step
  - http://www.springframework.org/docs/MVC-step-by-step/Spring-MVC-step-b
- Introduction to the Spring Framework
  - By Eduardo Isso Ito
    - http://www.springframework.org/downloads/EduardoIss
-


Form Controllers handling form interaction
Multipart Resolver to handle file uploads
Dynamic binding of data to the domain model
Integration with JSP, Velocity, XSLT, PDF, Excel
Spring Web MVC
WebApplicationContext providing e.g. messaging
Spring Web
Declarative transaction management for POJOs
Spring Context
Custom business logic
Sending Email
Remote access via Hession, Burlap, SOAP
Spring AOP
Spring ORM
Hibernate mappings
Custom Hibernate DAOs
Spring Core
Spring DAO
Servlet Container (Tomcat / Jetty)