**Spring Component:**

Core  
Context   
AOP   
ORM  
MVC

**Core**:

Provides functionality of framework. Core is implemenation of BeanFactory.

Bean factory applies IOC. IOC is a pattern that seperates configuration from actual source code

**Context** 🡪 ApplicationContext

Context is where the Spring beans are living

It is spring configuration file that provides context information and enterprise services to the framework

The ApplicationContext is where your Spring beans live.

**ContextLoaderListener**

The purpose of the ContextLoaderListener is two-fold:

To tie the lifecycle of the Spring ApplicationContext to the lifecycle of the ServletContext and

to automate the creation of the ApplicationContext, so you don't have to write explicit code to do create it - it's a convenience function.

Another convenient thing about the ServletContextListener is that it creates a WebApplicationContext and WebApplicationContext provides access to the ServletContext via ServletContextAware beans and the getServletContext method.

**ServletContext :**

The ServletContext is an object that contains meta information about your web application. we can set attributes in ServletContext and will be accessible via applications

**ServletConfig**

A Unique object per servlets. We cannot set attributes in ServletConfig.

-----------------------------------------------------------------------------------------------------------------------------

**Enterprise services:**

EJB   
Email  
Localization  
Validation 🡪 spring supports javax.validation.ConstraintValidator

Provies SpringBeanAutowiringInterceptor

@Stateless(mappedName = "PhotoSession")

@Interceptors(SpringBeanAutowiringInterceptor.class)

@TransactionAttribute(TransactionAttributeType.NOT\_SUPPORTED)

public class PhotoSession implements PhotoRemote {

@Autowired

private PhotoManager photoManager;

<beans :bean id="messageSource" class="org.springframework.context.support.ReloadableResourceBundleMessageSource">

@Size(min=2, max=30),  
 @Min(18),  
 @NotNull

Spring MVC supports JSR-303 specs by default we can add JSR-303 and it’s implementation in Spring MVC application. --> javax.validation.ConstraintValidator

**AOP** :

Aspect oriented programming

AOP can hijack the executing method, and add extra functionality before or after the method execution.

Modularizes individual pieces of logic using crosscutting implementation.

It separates business logics from system components

It’s just an interceptor to intercept some processes During the exexution of method

Has 2 types Static and Dynamic

**Joinpoint**

Are a core concept of AOP and define the points in your application at which we can insert additional logic

**Advice**

The code that is executed at a particular joint point, like before advice or after advice

**Pointcuts**

IS a collection of jointpoits that you use to define when advice should be executed

**Aspect**

It’s a combination of advice and pointcuts

**Weaving**

This is the process of actually inserting aspects in to the application code at the appropriate

Point  
AOP example

@Aspect

public class WebExceptionAspect {

@AfterThrowing(pointcut="execution(public \* ca.invesco.\*)") {

<bean name ="performnaceMonitorAdvice" class="ca.invesco.web.common.aop.service.PerformanceMonitorAspect"

<aop:aspectj-autoproxy proxy-target-class="true">

<aop:include name="performanceNonitorAdvice" />

<aop:aspectJ-autoproxy>

**ORM** ( Object Relational Map)

The Spring framework plugs into several ORM frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps

**MVC**

The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications.

The MVC framework is highly configurable via strategy interfaces and JSP, Velocity, Tiles, iText, and POI.

**Spring framework Benefit**

**Layer based architecture** :

We can just use one module and no need other modules like MVC

**Support AOP** :

Hijack the execution of method by adding functionality before and after the

method execution. cross cut implementation.

**Support IOC** :

Inversion of control is some set of technics to wire services and components to an

application.

Instead of application calling the framework it is the framework that

calls the components that are specified by application

Has **Dependency injection** :

Means giving a reference of one object to other.

Passing the service to the client

Dependency is an object that can be used (a service).

Injection is the passing of a dependency to a dependent object (a client)

Spring dependencies

We do not create an object we describe how they should be created. No JDNI

we describe which services are needed by which component in configuration file

IOC or Spring container is responsible for hooking it up from configuration fil

**Spring Dependency injection**

**Construction injection** provided by constructer Parameter  
<bean id="encyclopedia" name="encyclopedia-ref" class="ca.arkhls.ch03.listing36.model.ConfigurableEncyclopedia" >

<constructor-arg>  
 <util:map>  
 <entry key="AgeOfUniverse" value="100" />  
 <entry key="ConstantOfLife" value="1250" />  
 </util:map>  
 </constructor-arg>

**Setter injection** : provided by Java bean

<bean id="simple" class="ca.arkhls.ch03.listing36.model.Simple" init-method="init">

<property name="name" value=""/>

<property name="age" value="0"/>

</bean>

**Dependence lookup** : it is more coupled.

Lookup the JNDI to obtain dependecies from registration -->

ApplicationCOntext context = new XmlPathXmlApplicationContext(classpath\*:/spring-conifg.xml);

MyService myservice = Contenx.getBean("my service")

IOC injects the components dependencies in to the component via java bean.

**Register beans in Sring**

@Component – Indicates a auto scan component. Register a class in to SPring context.

@Repository – Specialization of @Component for DAO layer. Supports DAO exception

@Service – Specialization of @Component Indicates a Service component in the business layer.

@Controller – Indicates a controller component in the presentation layer.

http://www.programmingforliving.com/2012/10/how-to-write-custom-bean-scope-in-spring.html

**Bean instantiation** (Scope):

<bean id="..." class="..." scope="singleton">

By default all Beans are Singlton. we can change that to the following :

SINGLTON : Retuirn a single bean instance per Spring container

PROTOTYPE: Returns a new instance of bean each time we request

REQUEST : Returns a single bean instance per HttpRequest

SESSION : Returns a single bean instance per HttpSession

GLOBALSESSION: Return a single bean instance per global HTTP session. \*

THREAD : Scopes a single bean definition to thread local instance.

Every request for the bean will return the same instance with in a thread.

This scope is not registered to the container by default, if you want to use this, you will have to register this explicitly as given below

<bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">

<property name="scopes">  
 <map>  
 <entry key="thread">

<bean class="org.springframework.context.support.SimpleThreadScope"/>

</entry>

</map>

</property>

</bean>

**Spring Dependency check : @Required**

Dependency checking feature ensure the required properties have been set or injected.

NONE – No dependency checking.

SIMPLE – If any properties of primitive type (int, long,double…) and collection types (map, list..) have not been set, UnsatisfiedDependencyException will be thrown.

OBJECTS – If any properties of object type have not been set, UnsatisfiedDependencyException will be thrown.

ALL – If any properties of any type have not been set, an UnsatisfiedDependencyException

will be thrown.

**Bean Factory**   
It is a factory class that contains a collection of Beans. when we need any beans we can get it out of FactoryBean

**ApplicationContext:**

It is an extension to BeanFactory to manage the beans

**Localisation**

1. **Create ResourceBundleMessageSource**

<bean id="messageSource" class="org.springframework.context.support.ResourceBundleMessageSource">

<property name="basenames" >

<list>  
 <value>messages</value>

</list>

</property>

</bean>

1. **Create Locale change interceptor**

<bean id="localeChangeInterceptor" class="org.springframework.web.servlet.i18n.**LocaleChangeInterceptor**">

<property name="paramName" value="lang" />

</bean>

<bean class="org.springframework.web.servlet.mvc.annotation.**DefaultAnnotationHandlerMapping**">

<property name="interceptors">

<list>

<ref bean="localeChangeInterceptor" />

</list>

</property>

</bean>

**Change locale :**

we create Spring intercepter to intercept the request HandlerInterceptorAdaptor

HandlerInterceptorAdaptor : it is the class that we extends in our change Locale

**Lazy Initiation** (lazy-init)

Initilaize the beans when they are required is lazy loading. lazy-init is an attriute in IOC that tells the context not to initiate the beans at startup

**Autowireing bean**

Spring container is able to wire up relationships between collaborating beans automatically by inspecting the contents of the BeanFactory. The autowiring functionality has five modes.

**Component-Scan**

We can scan a particular package and its subpackages for annotated classes and autowairing.

<context: component-scan base-package=”ca.arkhls.service” />

By default, the @Autowired will perform the dependency checking to make sure the property has been wired properly.

Main context of web.xml

----------------

Here we can have 2 configuration for spring. root-context.xml at startup and servlet-context.xml file

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/root-context.xml</param-value>

</context-param>

<listener>

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

</listener>

------------------------------------------------------------------------

<servlet>

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

http://kumarnvm.blogspot.ca/2013/10/spring-task-execution-and-scheduling-25.html

Spring CustomScopeConfigurer ?

what is org.springframework.beans.factory.config.CustomScopeConfigurer ?

ScopeConfigurer module is a custom scope implementation for providing thread scoped beans.

Every request for a bean will return the same instance for the same thread.

A Runnable must be wrapped in a ThreadScopeRunnable if destruction callbacks should occur on a thread scoped bean.

2**. Scheduling fo Task**

Spring TimerManagerTaskScheduler ?

Spring also features integration classes for supporting scheduling with the Timer

Spring MVC : MODEL and VIEW

<servlet>

<servlet-name>appServlet</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>ContextConfigLocation</param-name>

<param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>appServlet</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<listener>

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

</listener>

**ApplicationContextAware** : To access Context in none spring classes

Spring MVC startup

InitializingBean interface to implement -->

@PostConstruct

@ExceptionHandler could return ModelAndView or JSON String

Model : attributes that we add to be viewed

The ModelAndView is simply a combination of the Model ( attributes that we added in the scope) and the view to be rendered by the Controller

Exposes several methods to add attributes to use in the view

ModelAndView mv = new ModelAndView(abc.jsp);

Model.addAttribute("persons",List<> pewrson);

ModelAttribute @ModelAttribute

It is biding a pojo inisde the model to Spring form

@ModelAttribute("user") User user;

BindingResult

It is an interface that extends Erros, we use BindingResult if sumbit form has not been validated

@Controller

public class UserController {

@RequestMapping(value="user", method = RequestMethod.GET)

public User user(){

return new User();

}

@RequestMapping(value="createUser", method = RequestMethod.POST)

public ModelAndView createUser(@ModelAttribute("user") @Valid User user,BindingResult result,ModelMap model) {

if(result.hasErrors()) {

return new ModelAndView("user");

}

model.addAttribute("name",user.getName());

model.addAttribute("age",user.getAge());

model.addAttribute("location",user.getLocation());

return new ModelAndView("redirect:pages/success.jsp");

}

}

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

import javax.validation.constraints.Size;

public class User {

@NotNull

@Size(min=5, max=20)

private String name;

}

Difference between Spring 3 and 4

1. Supports JDK 8

2. deprecated packages and methods have been removed

3. Added manage time zone support to localeContext

4. @Description annoation has been introducded to develoepr using Java-based configuration

5. spring-websocket module provides comprehensive support for WebSocket-based

6. new features in unit testing.

Spring Security :

Step 1

@EnableWebSecurity annotation and WebSecurityConfigurerAdapter work together to provide web based security.

By extending WebSecurityConfigurerAdapter

@Configuration

@EnableWebSecurity

public class HelloWebSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) {

auth

.inMemoryAuthentication()

.withUser("user").password("password").roles("USER");

}

}

Step 2

AbstractAnnotationConfigDispatcherServletInitializer --> to Register the Spring WebSecurity

Step 3

The last step is we need to map the springSecurityFilterChain.

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>ERROR</dispatcher>

<dispatcher>REQUEST</dispatcher>

</filter-mapping>

Difference between Sping boot and spring

Simplifies your Spring dependencies, no more version collisions

can be run straight from a command line without an application container

build more with less code - no need for XML, not even web.xml, auto-configuration

Spring Transactional attrubutes

PROPAGATION\_REQUIRED Support a current transaction, create a new one if none exists.This is the default setting.

If an existing transaction is in progress, method will run within that transaction, else a new transaction will be started.

PROPAGATION\_SUPPORTS Support a current transaction, execute non -transactionally if none exists.

PROPAGATION\_MANDATORY Support a current transaction, throw an exception if none exists.

PROPAGATION\_REQUIRES\_NEW Create a new transaction, suspend the current transaction if one exists

PROPAGATION\_NOT\_SUPPORTED Execute non - transactionally, suspend the current transaction if one exists.

PROPAGATION\_NEVER Execute non - transactionally, throw an exception if a transaction exists

PROPAGATION\_NESTED Method should run within a nested transaction if an existing transaction exists.The nested transaction can be committed/roll backed independently of the enclosing transaction. If no enclosing transaction behave like PROPAGATION\_REQUIRED.