Spring 3

André Faria Gomes

What?

The Spring Framework is a **lightweight** solution and a potential one-stop-shop for building your enterprise-ready applications. Spring Framework **is a Java platform** that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure **so you can focus on your application**.

Modules

Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
(MVC / Remoting)
Web
Servlet
Portlet
Struts
AOP
Aspects
Instrumentation
Core Container
Beans
Core
Context
Expression Language
Test

Dependency Managment

(not Dependency Injection)

- Maven
- OSGI
- Ivy

Modularization

- org.springframework.aop
- org.springframework.beans
- org.springframework.context
- org.springframework.context.support
- org.springframework.expression
- org.springframework.instrument
- org.springframework.jdbc
- org.springframework.jms
- org.springframework.orm
- org.springframework.oxm
- org.springframework.test
- org.springframework.transaction
- org.springframework.web
- org.springframework.web.portlet
- org.springframework.web.servlet
- org.springframework.web.struts

http://www.flickr.com/photos/9619972@N08/2781329487/

News

Java 5 Support

Early support for Java EE 6

Comprehensive REST support

Improved Documentation

http://www.flickr.com/photos/zen/7880592/sizes/o/

DI - Dependency Injection
(IoC - Inversion Of Control)

Dependencies

DI

Your Business Objects (POJOs)
Configuration Metadata
The Spring Container
produces
Fully configured system
Ready for Use

DI

BeanFactory
configuration framework and basic functionality
ApplicationContext
enterprise-specific functionality
WebApplicationContext
web functionality

DI

Constructor Based DI

Setter Based DI

BeanFactory

```
T getBean(Class<T> requiredType)

T getBean(String name, Class<T> requiredType)

Map<String, T> getBeansOfType(Class<T> type)
```

http://www.flickr.com/photos/durotriges/215248008/sizes/l/

Configuration

Types

- XML
- Annotations
- Java MetaData

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
  </bean>

 <bean id="petStore"
      class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">
    <property name="accountDao" ref="accountDao"/>
    <property name="itemDao" ref="itemDao"/>
    <!-- additional collaborators and configuration for this bean go here -->
  </bean>

</beans>
```

XML Composition

```
<beans>

    <import resource="services.xml"/>
    <import resource="resources/messageSource.xml"/>
    <import resource="/resources/themeSource.xml"/>

    <bean id="bean1" class="..."/>
    <bean id="bean2" class="..."/>

</beans>
```

The Bean Definition

Instanciation

```
<!-- Constructor -->

<bean id="bar" class="x.y.Bar"/>

<bean id="foo" class="x.y.Foo">
        <constructor-arg ref="bar"/>
        <constructor-arg ref="baz"/>
        <constructor-arg type="int" value="7500000"/>
</bean>

<!-- Factory Method -->
<bean id="exampleBean"
      class="examples.ExampleBean2"
      factory-method="createInstance"/>

<!-- Factory Object -->
<bean id="exampleBean"
      factory-bean="serviceLocator"
      factory-method="createInstance"/>
```

The P Namespace

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean name="classic" class="com.example.ExampleBean">
        <property name="email" value="foo@bar.com"/>
    </bean>

    <bean name="p-namespace" class="com.example.ExampleBean"
          p:email="foo@bar.com"/>
</beans>
```

Depends On

```
<bean id="beanOne" class="ExampleBean" depends-on="manager"/>
```

Used to Initialized and Destroy the Bean

Lazy Initialization

```
<bean id="lazy" class="com.foo.ExpensiveToCreateBean" lazy-init="true"/>

<beans default-lazy-init="true">
    <!-- no beans will be pre-instantiated... -->
</beans>
```

http://www.flickr.com/photos/hryckowian/2029393648/sizes/l/
Autowire

Autowire Types

- no (default)
- byName
- byType
- constructor (byType)
- autodetect

Autowiring Preferences

- You can exclude a bean from autowiring seting the optional property autowire-candidate to false.

- The top-level <beans/> element accepts one or more patterns within its default-autowire-candidates attribute

Checking for Dependencies

- If you want to check if all the dependencies were wired set the bean property dependency-check to:
  - simple: Dependency checking for primitives and collections.
  - object: Dependency checking for collaborators only.
  - all: Checking for collaborators, primitives, and collections.
  - none (default) No dependency checking.
- The Annotation @Required is an alternative to that.

http://www.flickr.com/photos/ingmar/3677214065/sizes/o/

Method Injection

Method Injection

In most application scenarios, most beans in the container are singletons. When a singleton bean needs to collaborate with another singleton bean, or a non-singleton bean needs to collaborate with another non-singleton bean, you typically handle the dependency by defining one bean as a property of the other. A problem arises when the bean lifecycles are different.

Awareness

```
public interface ApplicationContextAware {

    void setApplicationContext(ApplicationContext applicationContext) throws BeansException;
}
```

```
public interface BeanNameAware {

    void setBeanName(string name) throws BeansException;
}
```

Container Awareness

```
public class CommandManager implements ApplicationContextAware {

   private ApplicationContext applicationContext;

   public Object process(Map commandState) {
      // grab a new instance of the appropriate Command
      Command command = createCommand();
      // set the state on the (hopefully brand new) Command instance
      command.setState(commandState);
      return command.execute();
   }

   protected Command createCommand() {
      // notice the Spring API dependency!
      return this.applicationContext.getBean("command", Command.class);
   }

   public void setApplicationContext(ApplicationContext applicationContext)
      this.applicationContext = applicationContext;
   }
}
```

Lookup Method

```
// no more Spring imports!
public abstract class CommandManager {

   public Object process(Object commandState) {
      Command command = createCommand();
      command.setState(commandState);
      return command.execute();
   }

   protected abstract Command createCommand();
}

<bean id="command" class="fiona.apple.AsyncCommand" scope="prototype"/>

<!-- commandProcessor uses statefulCommandHelper -->
<bean id="commandManager" class="fiona.apple.CommandManager">
  <lookup-method name="createCommand" bean="command"/>
</bean>
```

http://www.flickr.com/photos/seandreilinger/1353110770/sizes/o/

Singleton

Only one instance is ever created...
<bean id="..." class="...">
<property name="accountDao"
ref="accountDao"/>
</bean>
<bean id="..." class="...">
<property name="accountDao"
ref="accountDao"/>
</bean>
<bean id="..." class="...">
<property name="accountDao"
ref="accountDao"/>
</bean>
1
<bean id="accountDao" class="..." />
... and this same shared instance is injected into each collaborating object

Prototype

Spring does not Manage the Prototype State After Creation

Web Scopes

Configuration for Web Scopes
(not required for Spring MVC)

```
<web-app>
...
<listener>
  <listener-class>
      org.springframework.web.context.request.RequestContextListener
  </listener-class>
</listener>
...
</web-app>
```

```
<bean
  id="loginAction"
  class="com.foo.LoginAction"
  scope="request"/>
```

Request

```
<bean
  id="userPreferences"
  class="com.foo.UserPreferences"
  scope="session"/>
```

Session

```
<bean id="userPreferences"
   class="com.foo.UserPreferences"
   scope="session">
     <aop:scoped-proxy/>
</bean>

<bean id="userManager"
   class="com.foo.UserManager">
   <property
      name="userPreferences"
      ref="userPreferences"/>
</bean>
```

Scoped Proxy

(only for web scopes)

http://www.flickr.com/photos/futurowoman/4103401582/sizes/o/

Custom Scopes

http://www.flickr.com/photos/29487767@N02/3592571491/sizes/l/

Bean LifeCycle Callbaks

LifeCycle CallBacks
by Interface

```
public class AnotherExampleBean implements InitializingBean {

    public void afterPropertiesSet() {
        // do some initialization work
    }
}


public class AnotherExampleBean implements DisposableBean {

    public void destroy() {
        // do some destruction work (like releasing pooled
connections)
    }
}
```

LifeCycle CallBacks
by Configuration

```
<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="cleanup"/>

<bean id="exampleInitBean" class="examples.ExampleBean" destroy-method="cleanup"/>
```

Default LifeCycle CallBacks

```
<beans default-init-method="init">

    <bean id="blogService" class="com.foo.DefaultBlogService">
        <property name="blogDao" ref="blogDao" />
    </bean>

</beans>
```

LifeCycle CallBacks
by Annotations

```
public class CachingMovieLister {

    @PostConstruct
    public void populateMovieCache() {
        // populates the movie cache upon
initialization...
    }

    @PreDestroy
    public void clearMovieCache() {
        // clears the movie cache upon destruction...
    }
}
```

Spring Lifecycle CallBack by Interface

```
public interface Lifecycle {

    void start();

    void stop();

    boolean isRunning();

}
```

```
public interface LifecycleProcessor extends Lifecycle {

    void onRefresh(); //Refrers to the Context

    void onClose(); //Refrers to the Context

}
```

http://www.flickr.com/photos/flatcat/3555774083/

Bean PostProcessor

Bean PostProcessor

- The BeanPostProcessor interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic, and so forth.
- You might extend the Spring IoC container. An example is the @Required Annotation

Bean PostProcessor

```
public class InstantiationTracingBeanPostProcessor implements BeanPostProcessor {

    // simply return the instantiated bean as-is
    public Object postProcessBeforeInitialization(Object bean, String beanName)
                                                                    throws BeansException {
        return bean; // we could potentially return any object reference here...
    }

    public Object postProcessAfterInitialization(Object bean, String beanName)
                                                                    throws BeansException {
        return bean;
    }
}
```

http://www.flickr.com/photos/samiksha/4187205767/

Bean Definition Inheritance

```
<bean id="inheritedTestBean" abstract="true"
    class="org.springframework.beans.TestBean">
  <property name="name" value="parent"/>
  <property name="age" value="1"/>
</bean>

<bean id="inheritsWithDifferentClass"
      class="org.springframework.beans.DerivedTestBean"
      parent="inheritedTestBean" init-method="initialize">

  <property name="name" value="override"/>
  <!-- the age property value of 1 will be inherited from  parent -->

</bean>
```

Bean Definition Inheritance

```
<bean id="inheritedTestBeanWithoutClass" abstract="true">
    <property name="name" value="parent"/>
    <property name="age" value="1"/>
</bean>

<bean id="inheritsWithClass" class="org.springframework.beans.DerivedTestBean"
    parent="inheritedTestBeanWithoutClass" init-method="initialize">
  <property name="name" value="override"/>
  <!-- age will inherit the value of 1 from the parent bean definition-->
</bean>
```

Abstract Bean Definition Inheritance

Annotation Based Configuration

No XML

```
<context:annotation-config/>
```

@Required

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Required
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }

    // ...
}
```

@Autowired ou @Inject

```
public class MovieRecommender {

    private MovieCatalog movieCatalog;

    private CustomerPreferenceDao customerPreferenceDao;

    @Autowired
    public void prepare(MovieCatalog movieCatalog,
                        CustomerPreferenceDao customerPreferenceDao) {
        this.movieCatalog = movieCatalog;
        this.customerPreferenceDao = customerPreferenceDao;
    }

    // ...
}
```

@Autowired All Beans of a Type

```
public class MovieRecommender {

    @Autowired(required=false) //true is the default
    private MovieCatalog[] movieCatalogs;

    @Autowired
    private Set<MovieCatalog> movieCatalogs2;

    @Autowired //Map<BeanName, Bean>
    private Map<String, MovieCatalog> movieCatalogsMap;
}
```

Automatic Autowired

You can also use @Autowired for interfaces that are well-known resolvable dependencies:

BeanFactory,
ApplicationContext,
ResourceLoader,
ApplicationEventPublisher,
MessageSource.

@Resource

```
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Resource(name="myMovieFinder")
    public void setMovieFinder(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }
}
```

2° byType

1° byName

```
public class MovieRecommender {

    @Resource
    private CustomerPreferenceDao customerPreferenceDao;

    @Resource
    private ApplicationContext context;

}
```

Qualifiers

Because autowiring by type may lead to multiple candidates, it is often necessary to have more control over the selection process. Even though qualifiers do not have to be unique.

Qualifier

```java
public class MovieRecommender {

    @Autowired
    @Qualifier("main")
    private MovieCatalog movieCatalog;

    // ...
}
```

```xml
<bean class="example.SimpleMovieCatalog">
    <qualifier value="main"/>
</bean>

<bean class="example.SimpleMovieCatalog">
    <qualifier value="action"/>
</bean>
```

Qualifier

```java
@Target({ElementType.FIELD, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Genre {
    String value();
}
```

```xml
<bean class="example.SimpleMovieCatalog">
        <qualifier
          type="example.Genre"
          value="Comedy"/>
</bean>
```

```java
public class MovieRecommender {

    @Autowired
    @Genre("Action")
    private MovieCatalog actionCatalog;

    private MovieCatalog comedyCatalog;

    @Autowired
    public void setComedyCatalog(@Genre("Comedy") MovieCatalog comedyCatalog) {
        this.comedyCatalog = comedyCatalog;
    }

    // ...
}
```

Qualifier

```
public class MovieRecommender {

    @Autowired
    @MovieQualifier(format=Format.VHS, genre="Action")
    private MovieCatalog actionVhsCatalog;

    @Autowired
    @MovieQualifier(format=Format.DVD, genre="Action")
    private MovieCatalog actionDvdCatalog;

}
```

```
<bean class="example.SimpleMovieCatalog">
    <qualifier type="MovieQualifier">
        <attribute key="format" value="VHS"/>
        <attribute key="genre" value="Action"/>
    </qualifier>
</bean>

<bean class="example.SimpleMovieCatalog">
    <meta key="format" value="DVD"/>
    <meta key="genre" value="Action"/>
</bean>
```

CustomAutowireConfigurer

```
<bean id="customAutowireConfigurer"
       class="org.springframework.beans.factory.annotation.CustomAutowireConfigurer">
    <property name="customQualifierTypes">
        <set>
            <value>example.CustomQualifier</value>
        </set>
    </property>
</bean>
```

A BeanFactoryPostProcessor that enables you to register your own custom qualifier annotation types even if they are not annotated with Spring's @Qualifier annotation

Stereotype Annotations

You can annotate your component classes with@Component, but by annotating them with @Repository, @Service, or @Controller instead, your classes are more properly suited for processing by tools or associating with aspects.

Stereotype

```
@Service
public class SimpleMovieLister {

    private MovieFinder movieFinder;

    @Autowired
    public SimpleMovieLister(MovieFinder movieFinder) {
        this.movieFinder = movieFinder;
    }
}

@Repository
public class JpaMovieFinder implements MovieFinder {}
```

Stereotype Scanners

```
<context:component-scan base-package="org.example"/>
```

Annotation	org.example.SomeAnnotation	An annotation to be present at the type level in target components.
assignable	org.example.SomeClass	A class (or interface) that the target components are assignable to (extend/ implement).
aspectj	org.example..*Service+	An AspectJ type expression to be matched by the target components.
regex	org\.example\.Default.*	A regex expression to be matched by the target components class names.
custom	org.example.MyTypeFilter	A custom implementation of the org.springframework.core.type .**TypeFilter** interface.

Stereotype Scanners

```
<beans>

    <context:component-scan base-package="org.example">
       <context:include-filter type="regex" expression=".*Stub.*Repository"/>
       <context:exclude-filter type="annotation"

expression="org.springframework.stereotype.Repository"/>
    </context:component-scan>

</beans>
```

Defining Bean MetaData in Components

```
@Component
public class FactoryMethodComponent {

    @Bean @Qualifier("public")
    public TestBean publicInstance() {
        return new TestBean("publicInstance");
    }

    public void doWork() {
        // Component method implementation omitted
    }
}
```

This class is a Spring component that has application-specific code contained in its doWork method. However, it also contributes a bean definition that has a factory method referring to the method publicInstance. The @Bean annotation identifies the factory method and other bean definition properties, such as a qualifier value through the @Qualifier annotation. Other method level annotations that can be specified are @Scope, @Lazy, and custom qualifier annotations.

Defining Bean MetaData in Components with more Complexity

```
@Component
public class FactoryMethodComponent {

    private static int i;

    @Bean @Qualifier("public")
    public TestBean publicInstance() {
        return new TestBean("publicInstance");
    }

    @Bean @BeanAge(1)
    protected TestBean protectedInstance(@Qualifier("public") TestBean spouse,
                                         @Value("#{privateInstance.age}") String country) {
        TestBean tb = new TestBean("protectedInstance", 1);
        tb.setSpouse(tb);
        tb.setCountry(country);
        return tb;
    }

    @Bean @Scope(BeanDefinition.SCOPE_SINGLETON)
    private TestBean privateInstance() {
        return new TestBean("privateInstance", i++);
    }

    @Bean @Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode = ScopedProxyMode.TARGET_CLASS)
    public TestBean requestScopedInstance() {
        return new TestBean("requestScopedInstance", 3);
    }
}
```

Naming Generation

```
@Service("myMovieLister")
public class SimpleMovieLister {}

@Repository
public class MovieFinderImpl implements MovieFinder {}

<beans>

    <context:component-scan base-package="org.example"
                            name-generator="org.example.MyNameGenerator" />

</beans>
```

If want to define a strategy to name beans,
you might implement a BeanNameGenerator

@Configuration

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

equivalent to

```
<beans>
    <bean id="myService" class="com.acme.services.MyServiceImpl"/>
</beans>
```

Config Import

```
@Configuration
public class ConfigA {
    public @Bean A a() { return new A(); }
}

@Configuration
@Import(ConfigA.class)
public class ConfigB {
    public @Bean B b() { return new B(); }
}
```

```
ApplicationContext ctx = new AnnotationConfigApplicationContext(ConfigB.class);
// now both beans A and B will be available...
A a = ctx.getBean(A.class);
B b = ctx.getBean(B.class);
```

equivalent to </import> in the xml

Dependencies

```
@Configuration
public class ServiceConfig {
    private @Autowired AccountRepository accountRepository;

    public @Bean TransferService transferService() {
        return new TransferServiceImpl(accountRepository);
    }
}

@Configuration
public class RepositoryConfig {
    private @Autowired DataSource dataSource;

    public @Bean AccountRepository accountRepository() {
        return new JdbcAccountRepository(dataSource);
    }
}

@Configuration
@Import({ServiceConfig.class, RepositoryConfig.class})
public class SystemTestConfig {
    public @Bean DataSource dataSource() { /* return new DataSource */ }
}

public static void main(String[] args) {
    ApplicationContext ctx = new AnnotationConfigApplicationContext(SystemTestConfig.class);
    // everything wires up across configuration classes...
    TransferService transferService = ctx.getBean(TransferService.class);
    transferService.transfer(100.00, "A123", "C456");
}
```

Combine Java and XML

```
@Configuration
@ImportResource("classpath:/com/acme/properties-config.xml")
public class AppConfig {
    private @Value("${jdbcProperties.url}") String url;
    private @Value("${jdbcProperties.username}") String username;
    private @Value("${jdbcProperties.password}") String password;

    public @Bean DataSource dataSource() {
        return new DriverManagerDataSource(url, username, password);
    }
}
```

Combine Java and XML

```java
@Configuration
public class AppConfig {
    private @Autowired DataSource dataSource;

    public @Bean AccountRepository accountRepository() {
        return new JdbcAccountRepository(dataSource);
    }

    public @Bean TransferService transferService() {
        return new TransferService(accountRepository());
    }
}
```

```xml
<beans>
    <!-- enable processing of annotations such as @Autowired and @Configuration -->
    <context:annotation-config/>
    <context:property-placeholder location="classpath:/com/acme/jdbc.properties"/>

    <bean class="com.acme.AppConfig"/>

    <bean class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>
</beans>
```

Lifecycle (Option 1)

```
@Configuration
public class AppConfig {
    @Bean(initMethod = "init")
    public Foo foo() {
        return new Foo();
    }
    @Bean(destroyMethod = "cleanup")
    public Bar bar() {
        return new Bar();
    }
}
```

Lifecycle (Option 2)
You Call It

```
@Configuration
public class AppConfig {
    @Bean
    public Foo foo() {
        Foo foo = new Foo();
        foo.init();
        return foo;
    }
}
```

Java Based Metadata

- @Configuration
- @Bean
- @DependsOn
- @Primary
- @Lazy
- @Import
- @ImportResource
- @Value

Java Based Metadata

```
<context:component-scan
  base-package="org.example.config"/>

<util:properties
  id="jdbcProperties"
  location="classpath:org/example/config/jdbc.properties"/>
```

```java
package org.example.config;

@Configuration
public class AppConfig {
    private @Value("#{jdbcProperties.url}") String jdbcUrl;
    private @Value("#{jdbcProperties.username}") String username;
    private @Value("#{jdbcProperties.password}") String password;

    @Bean
    public FooService fooService() {
        return new FooServiceImpl(fooRepository());
    }

    @Bean
    public FooRepository fooRepository() {
        return new HibernateFooRepository(sessionFactory());
    }

    @Bean
    public SessionFactory sessionFactory() {
        // wire up a session factory
        AnnotationSessionFactoryBean asFactoryBean =
            new AnnotationSessionFactoryBean();
        asFactoryBean.setDataSource(dataSource());
        // additional config
        return asFactoryBean.getObject();
    }

    @Bean
    public DataSource dataSource() {
        return new DriverManagerDataSource(jdbcUrl, username, password);
    }
}
```

Instantianting the Container

```
public static void main(String[] args) {

    ApplicationContext ctx =
      new AnnotationConfigApplicationContext(AppConfig.class);

    ApplicationContext ctx =
     new ClassPathXmlApplicationContext(
       new String[] {"services.xml", "daos.xml"});


     FooService fooService = ctx.getBean(FooService.class);

     fooService.doStuff();

}
```

Medata in web.xml

```
<context-param>
    <param-name>contextClass</param-name>
     <param-value>
        org.springframework.web.context.support.AnnotationConfigWebApplicationContext
    </param-value>
</context-param>
```

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

http://www.flickr.com/photos/natura_pagana/3802901392/

Application Context Events

ContextRefreshedEvent

Published when the ApplicationContext is initialized or refreshed, for example, using the refresh() method on theConfigurableApplicationContext interface. "Initialized" here means that all beans are loaded, post-processor beans are detected and activated, singletons are pre-instantiated, and the ApplicationContext object is ready for use. As long as the context has not been closed, a refresh can be triggered multiple times, provided that the chosen ApplicationContext actually supports such "hot" refreshes. For example, XmlWebApplicationContext supports hot refreshes, but GenericApplicationContext does not.

Other Events

- ContextStartedEvent
- ContextStoppedEvent
- ContextClosedEvent
  - when it cannot be refreshed or restarted
- RequestHandledEvent
  - A web-specific event telling all beans that an HTTP request has been serviced. This event is published after the request is complete.

Event Example (1)

```
<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"/>
</bean>
```

Event Example (2)

```
public class EmailBean implements ApplicationContextAware {

    private List blackList;
    private ApplicationContext ctx;

    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 event = new BlackListEvent(address, text);
            ctx.publishEvent(event);
            return;
        }
        // send email...
    }
}
```

Event Example (3)

```
public class BlackListNotifier implements ApplicationListener {

    private String notificationAddress;

    public void setNotificationAddress(String notificationAddress) {
        this.notificationAddress = notificationAddress;
    }

    public void onApplicationEvent(ApplicationEvent event) {
        if (event instanceof BlackListEvent) {
            // notify appropriate person...
        }
    }
}
```

http://www.flickr.com/photos/jeffwerner/329401643/sizes/l/

Resources

Spring Make it Easier!

The Spring's Resource Interface

```
public interface Resource extends InputStreamSource {

    boolean exists();

    boolean isOpen();

    URL getURL() throws IOException;

    File getFile() throws IOException;

    Resource createRelative(String relativePath) throws IOException;

    String getFilename();

    String getDescription();
}
```

Types of Resources

ClassPathResource
FileSystemResource
InputStreamResource
ByteArrayResource
ServletContextResource

```
var r = "classpath:some/resource/path/myTemplate.txt”;

Resource template = ctx.getResource(r);
```

```
Classpath: classpath:com/myapp/config.xml
File: file:/data/config.xml
Http: http://myserver/logo.png
None: /data/config.xml
```

Resource Loader

```
public interface ResourceLoader {
    Resource getResource(String location);
}
```

```
var r = "classpath:some/resource/path/myTemplate.txt”;

Resource template = resourceLoader.getResource(r);
```

The ApplicationContext is a ResourceLoader

Resource as Dependecies

```
<bean id="myBean" class="...">
  <property name="template" value="some/resource/path/myTemplate.txt"/>
</bean>

<property name="template" value="classpath:some/resource/path/myTemplate.txt">

<property name="template" value="file:/some/resource/path/myTemplate.txt"/>
```

Wildcard Ant-Style

```
/WEB-INF/*-context.xml
com/mycompany/**/applicationContext.xml
file:C:/some/path/*-context.xml
classpath:com/mycompany/**/applicationContext.xml
```

```
classpath*:META-INF/*-beans.xml
```

Conversion, Formatting and Validation

Converters

```
package org.springframework.core.convert.converter;

public interface Converter<S, T> {

    T convert(S source);

}

final class StringToInteger implements Converter<String, Integer> {

    public Integer convert(String source) {
        return Integer.valueOf(source);
    }

}
```

Formatters

```
package org.springframework.format;

public interface Formatter<T> extends Printer<T>, Parser<T> {
}

public interface Printer<T> {
    String print(T fieldValue, Locale locale);
}

import java.text.ParseException;

public interface Parser<T> {
    T parse(String clientValue, Locale locale) throws
}
```

Date Formatter

```
public final class DateFormatter implements Formatter<Date> {

    private String pattern;

    public DateFormatter(String pattern) {
        this.pattern = pattern;
    }

    public String print(Date date, Locale locale) {
        if (date == null) {
            return "";
        }
        return getDateFormat(locale).format(date);
    }

    public Date parse(String formatted, Locale locale) throws ParseException {
        if (formatted.length() == 0) {
            return null;
        }
        return getDateFormat(locale).parse(formatted);
    }

    protected DateFormat getDateFormat(Locale locale) {
        DateFormat dateFormat = new SimpleDateFormat(this.pattern, locale);
        dateFormat.setLenient(false);
        return dateFormat;
    }

}
```

Annotation Driven Formatter

```
public final class NumberFormatAnnotationFormatterFactory implements AnnotationFormatterFactory<NumberFormat> {

    public Set<Class<?>> getFieldTypes() {
        return new HashSet<Class<?>>(asList(new Class<?>[] {
            Short.class, Integer.class, Long.class, Float.class, Double.class, BigDecimal.class,
BigInteger.class }));
    }

    public Printer<Number> getPrinter(NumberFormat annotation, Class<?> fieldType) {
        return configureFormatterFrom(annotation, fieldType);
    }

    public Parser<Number> getParser(NumberFormat annotation, Class<?> fieldType) {
        return configureFormatterFrom(annotation, fieldType);
    }

    private Formatter<Number> configureFormatterFrom(NumberFormat annotation, Class<?> fieldType) {
        if (!annotation.pattern().isEmpty()) {
            return new NumberFormatter(annotation.pattern());
        } else {
            Style style = annotation.style();
            if (style == Style.PERCENT) {
                return new PercentFormatter();
            } else if (style == Style.CURRENCY) {
                return new CurrencyFormatter();
            } else {
                return new NumberFormatter();
            }
        }
    }
}
```

Annotation Driven Formatter

```
public class MyModel {

    @NumberFormat(style=Style.CURRENCY)
    private BigDecimal decimal;

}
```

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

Validation

JSR 303 Full Support

```
public class PersonForm {

    @NotNull
    @Size(max=64)
    private String name;

    @Min(0)
    private int age;

}
```

Validation

JSR 303 (no Spring Here)

```
private static Validator validator;

@BeforeClass
public static void before() {
    final ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
    validator = factory.getValidator();
}

    @Test
    public void testNomeIsNull() {
        final PersonForm personForm = new PersonForm();
        personForm.setNome(null);
        personForm.setIdade(10);

        final Set<ConstraintViolation< PersonForm>> violations = validator.validate(personForm);
        assertEquals(1, violations.size());
        assertEquals("may not be null", violations.iterator().next().getMessage());
    }
}
```

Validation

JSR 303 Full Support

```
import javax.validation.Validator;

@Service
public class MyService {

    @Autowired
    private Validator validator;
}
```

org.springframework.validation.Validator
or
javax.validation.Validator

```
<bean id="validator" class="org.springframework.validation.beanvalidation.LocalValidatorFactoryBean" />
```

Validation

in Spring MVC 3

```
@Controller
public class MyController {

    @RequestMapping("/foo", method=RequestMethod.POST)
    public void processFoo(@Valid Foo foo) { /* ... */ }
```

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

enable validation
across all controllers

http://www.flickr.com/photos/shawnecono/149172094/sizes/l/
anguage
Spring Expression Language

“The Spring Expression Language (SpEL) was created to provide the Spring community a single, well supported expression language that can be used across all the products in the Spring portfolio”

Other ELs

- OGNL
- MVEL
- JBoss EL

SpEL Feature Overview

- Literal expressions
- Boolean and relational operators
- Regular expressions
- Class expressions
- Accessing properties, arrays, lists, maps
- Method invocation
- Relational operators
- Assignment
- Calling constructors
- Ternary operator
- Variables
- User defined functions
- Collection projection
- Collection selection
- Templated expressions

Simple Evaluations

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Hello World'");
String message = (String) exp.getValue();

Expression exp = parser.parseExpression("'Hello World'.concat('!')"); //Hello World!

Expression exp = parser.parseExpression("'Hello World'.bytes");
byte[] bytes = (byte[]) exp.getValue();

Expression exp = parser.parseExpression("'Hello World'.bytes.length");
int length = (Integer) exp.getValue();

Expression exp = parser.parseExpression("name == 'Nikola Tesla'");
boolean result = exp.getValue(context, Boolean.class);  // evaluates to true
```

Evaluation with Specific Objects

```
/ Create and set a calendar
GregorianCalendar c = new GregorianCalendar();
c.set(1856, 7, 9);

//  The constructor arguments are name, birthday, and nationality.
Inventor tesla = new Inventor("Nikola Tesla", c.getTime(), "Serbian");

ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("name");

String name = (String) exp.getValue(tesla);
```

Evaluation with Root Object

```
// Create and set a calendar
GregorianCalendar c = new GregorianCalendar();
c.set(1856, 7, 9);

//  The constructor arguments are name, birthday, and nationality.
Inventor tesla = new Inventor("Nikola Tesla", c.getTime(), "Serbian");

ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("name");

EvaluationContext context = new StandardEvaluationContext();
context.setRootObject(tesla);

String name = (String) exp.getValue(context);
```

make it faster when called more than once caching java.lang.reflect's Method, Field, and Constructor instances

The Evaluation Context

- set the root object
- set variables
- set register functions

- extend the SpEL registering custom
  - ConstructorResolvers
  - MethodResolvers
  - PropertyAccessors

SpEL

```
<bean class="mycompany.RewardsTestDatabase">
    <property name="databaseName"
            value="#{systemProperties.databaseName}"/>

    <property name="keyGenerator"
            value="#{strategyBean.databaseKeyGenerator}"/>
</bean>

<bean id="taxCalculator" class="org.spring.samples.TaxCalculator">
    <property name="defaultLocale" value="#{ systemProperties['user.region'] }"/>
</bean>

<bean id="numberGuess" class="org.spring.samples.NumberGuess">
    <property name="randomNumber" value="#{ T(java.lang.Math).random() * 100.0 }"/>
</bean>
```

SpEL

```
@Repository
public class RewardsTestDatabase {

    @Value("#{systemProperties.databaseName}")
    public void setDatabaseName(String dbName) { … }

    @Value("#{strategyBean.databaseKeyGenerator}")
    public void setKeyGenerator(KeyGenerator kg) { … }
}
```

SpEL General Usage

```
// Arrays and Lists
Officers[0].PlaceOfBirth.City

// Maps
Officers['president']

// Methods
'abc'.substring(2, 3)
isMember('Mihajlo Pupin')

// Operators
2 == 2
2 < -5.0
black' < 'block
// Operators instanceof and Regex also suported
// ('<'), gt ('>'), le ('<='), ge ('>='), eq ('=='),
// ne ('!='), div ('/'), mod ('%'), not ('!')

// Assignment
Name = 'Alexandar Seovic'

// Constructor
new org.spring.samples.spel.inventor.Inventor('Albert Einstein','German)
```

SpEL Variables

```
Inventor tesla = new Inventor("Nikola Tesla", "Serbian");
StandardEvaluationContext context = new StandardEvaluationContext(tesla);
context.setVariable("newName", "Mike Tesla");

parser.parseExpression("Name = #newName").getValue(context);

System.out.println(tesla.getName()) // "Mike Tesla"
```

SpEL More Features

- Ternary Operador
  - x == y ? a : b
- Elvis Operador
  - name?:' Unknown'
- Safe Navigation
  - PlaceOfBirth?.City
- Collection Selection
  - Members.?[Nationality == 'Serbian']
  - map.?[value<27]
- Collection Projection
  - Members.![placeOfBirth.city] //new collection of places of birth

Expression Templates

```
String randomPhrase =
   parser.parseExpression("random number is ${T(java.lang.Math).random()}",
                          new TemplatedParserContext()).getValue(String.class);

// evaluates to "random number is 0.7038186818312008"
```

reduce the expressions
to only inside the ${}

http://www.flickr.com/photos/spodzone/3294241163/

Aspect

A modularization of a concern that cuts across multiple classes.

Transaction management is a good example.

Two Options

- XML Schema-based
- AspectJ Annotations Style (Java 5+)

Spring AOP or Full AspectJ?

introduce the AspectJ compiler / weaver into your development and build processes

JoinPoint

A point during the execution of a program, such as the execution of a method or the handling of an exception.

Advice

Action taken by an aspect at a particular join point.

Different types of advice include:

around
before
after returning
after throwing
After (finally)

In Sprint the advice is an interceptor, it mantains a chain of interceptors around the join point

Pointcut

A predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name).

Spring uses the AspectJ pointcut expression language by default.

Introduction or Inter-Type

Declaring additional methods or fields on behalf of a type.

For example, you could use an introduction to make a bean implement an IsModified interface, to simplify caching.

Target Object

An object being advised by one or more aspects.

AOP Proxy

An object created by the AOP framework in order to implement the aspect contracts (advise method executions and so on).

In the Spring Framework, an AOP proxy will be a JDK dynamic proxy or a CGLIB proxy.

Weaving

Linking aspects with other application types or objects to create an advised object.

This can be done at compile time (using the AspectJ compiler, for example), load time, or at runtime.

Spring AOP, like other pure Java AOP frameworks, performs weaving at runtime.

Considerations

Field interception is not implemented, although support for field interception could be added using AspectJ. The same for protected/private methods and constructors.

Enabling AspectJ Support

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

```
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class NotVeryUsefulAspect {}
```

Declaring a PointCut

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

```
@Pointcut("execution(* transfer(..))")// the pointcut expression
private void anyOldTransfer() {}// the pointcut signature
```

Supported Pointcuts Designators

- **execution** - for matching method execution
- **within** - limits matching to join points within certain types
- **this** - limits matching to join points where the bean reference (proxy) is an instance of the given type
- **target** - limits matching to join points where the target object is an instance of the given type
- **args** - limits matching to join points where the arguments are instances of the given types

Supported Pointcuts Designators

- **@target** - limits matching to join points where the class of the executing object has an annotation of the given type
- **@args** - limits matching to join points where the runtime type of the actual arguments passed have annotations of the given type(s)
- **@within** - limits matching to join points within types that have the given annotation
- **@annotation** - limits matching to join points where the subject of the join point (method) has the given annotation

Spring Pointcuts Designators

- **bean** - select and particular Spring Bean
  - bean(idOrNameOfBean)

Pointcut Expressions

```
execution(modifiers-pattern? ret-type-pattern declaring-
type-pattern? name-pattern(param-pattern) throws-pattern?)
```

```
@Pointcut("execution(public * *(..))")
private void anyPublicOperation() {}

@Pointcut("within(com.xyz.someapp.trading..*)")
private void inTrading() {}

@Pointcut("anyPublicOperation() && inTrading()")
private void tradingOperation() {}

@Pointcut("execution(* com.xyz.someapp.dao.*.*(..))")
public void dataAccessOperation() {}

@Pointcut("execution(* com.xyz.someapp.service.*.*(..))")
public void businessService() {}
```

Pointcut Expressions

```
execution(modifiers-pattern? ret-type-pattern declaring-
type-pattern? name-pattern(param-pattern) throws-pattern?)
```

- `execution(* set*(..))`
- `execution(* com.xyz.service.AccountService.*(..))`
- `this(com.xyz.service.AccountService)`
- `target(com.xyz.service.AccountService)`
- `args(java.io.Serializable)`
- `@target(org.springframework.transaction.annotation.Transactional)`
- `@within(org.springframework.transaction.annotation.Transactional)`
- `@annotation(org.springframework.transaction.annotation.Transactional)`
- `@args(com.xyz.security.Classified)`
- `bean(*Service)`

Designators

A well written pointcut should try and include at least the kinded and scoping designators

During compilation, AspectJ processes pointcuts in order to try and optimize matching performance.

- **Kinded designators** are those which select a particular kind of join point.
  - For example: execution, get, set, call, handler

- **Scoping designators** are those which select a group of join points of interest (of probably many kinds).
  - For example: within, withincode

- **Contextual designators** are those that match (and optionally bind) based on context.
  - For example: this, target, @annotation

Using Advices

```
@Aspect public class BeforeExample {
  @Before("execution(* com.xyz.myapp.dao.*.*(..))")
  public void doAccessCheck() {}
```

```
@AfterReturning(
   pointcut="com.xyz.myapp.SystemArchitecture.dataAccessOperation()",
   returning="retVal")
 public void doAccessCheck(Object retVal) {}

@AfterThrowing(
   pointcut="com.xyz.myapp.SystemArchitecture.dataAccessOperation()",
   throwing="ex")
 public void doRecoveryActions(DataAccessException ex) {}

@After("com.xyz.myapp.SystemArchitecture.dataAccessOperation()")
 public void doReleaseLock() {}

@Around("com.xyz.myapp.SystemArchitecture.businessService()")
 public Object doBasicProfiling(ProceedingJoinPoint pjp) throws Throwable {
   Object retVal = pjp.proceed(); // start stopwatch
   return retVal; // stop stopwatch
 }
```

Using Advices with Annotations

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Auditable {
   AuditCode value();
}

@Before("com.xyz.lib.Pointcuts.anyPublicMethod() && " +
        "@annotation(auditable)")
public void audit(Auditable auditable) {
  AuditCode code = auditable.value();
}
```

Using Advices with Parameters

```
@Pointcut("com.xyz.myapp.SystemArchitecture.dataAccessOperation() &&" +
          "args(account,..)")
private void accountDataAccessOperation(Account account) {}


@Before("accountDataAccessOperation(account)")
public void validateAccount(Account account) {}
```

Domain Objects DI with AspectJ Compiler/Weaver

Domain Objects DI with AOP

The **spring-aspects.jar** contains an annotation-driven aspect that exploits this capability to allow dependency injection of any object.

The support is intended to be used for objects created outside of the control of any container.

Domain objects often fall into this category because they are often created programmatically using the **new operator, or by an ORM tool as a result of a database query**.

The @Configurable Annotation

```
package com.xyz.myapp.domain;

import org.springframework.beans.factory.annotation.Configurable;

@Configurable
public class Account {

@Autowired
...
}
```

For this to work the annotated types must be woven with the AspectJ weaver in build-time or load-time

```
<context:spring-configured/>
```

Load Time Weaving (LTW)

Load-time weaving (LTW) refers to the process of weaving AspectJ aspects into an application's class files as they are being loaded into the Java virtual machine (JVM).

With the AspectJ LTW you can also call @Transaction annotateded methods in the same class, and it's goint to intercept

Setting the AspectJ Java Agent in the VM for Load Time Weaving (LTW)

http://www.flickr.com/photos/andresrueda/3274955487/sizes/l/

Transactions

Transações

GLOBAL (JPA)

Suporta Diversos Recursos

Bancos de Dados + Fila de Mensagens

LOCAL

Suporta apenas um recurso em específico

Transactions

ISOLATION

PROPAGATION

TIME OUT

READ ONLY STATUS

Transaction Managers

org.springframework.orm.hibernate3.HibernateTransactionManager

org.springframework.transaction.jta.JtaTransactionManager

Spring Transactions

XML Configuration (Optional)

```
<bean id="fooService" class="x.y.service.DefaultFooService"/>

<tx:advice id="txAdvice" transaction-manager="txManager">

<tx:attributes>
  <!-- all methods starting with 'get' are read-only -->
  <tx:method name="get*" read-only="true"/>
  <!-- other methods use the default transaction settings (see below) -->
  <tx:method name="*"/>
</tx:attributes>
</tx:advice>

<aop:config>
<aop:pointcut id="fooServiceOperation" expression="execution(*
x.y.service.FooService.*(..))"/>
<aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
```

Exceptions

In its default configuration, the Spring Framework's transaction infrastructure code *only* marks a transaction for rollback in the case of runtime, unchecked exceptions;

```
<tx:advice id="txAdvice" transaction-manager="txManager">
  <tx:attributes>
  <tx:method name="get*" read-only="true" rollback-for="NoProductInStockException"/>
  <tx:method name="*"/>
  </tx:attributes>
```

```
<tx:advice id="txAdvice">
  <tx:attributes>
  <tx:method name="updateStock" no-rollback-for="InstrumentNotFoundException"/>
  <tx:method name="*"/>
  </tx:attributes>
</tx:advice>
```

<tx:advice/> default settings

- Propagation setting is REQUIRED.
- Isolation level is DEFAULT.
- Transaction is read/write.
- Transaction timeout defaults to the default timeout of the underlying transaction system, or none if timeouts are not supported.
- Any RuntimeException triggers rollback, and any checked Exception does not.

@Transactional

```
// the service class that we want to make transactional
@Transactional
public class DefaultFooService implements FooService {}
```

```
<tx:annotation-driven transaction-manager="txManager"/>

<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager"/>
  <property name="dataSource" ref="dataSource"/>
</bean>
```

Spring recommends that you only annotate concrete classes (and methods of concrete classes) with the @Transactional

Multiple Transaction Managers

```
public class TransactionalService {

  @Transactional("order")
  public void setSomething(String name) { ... }

  @Transactional("account")
  public void doSomething() { ... }
}
```

```
<tx:annotation-driven/>

<bean id="transactionManager1" class="org.springframework.jdbc.DataSourceTransactionManager">
  ...
  <qualifier value="order"/>
</bean>

<bean id="transactionManager2" class="org.springframework.jdbc.DataSourceTransactionManager">
  ...
  <qualifier value="account"/>
</bean>
```

Custom TX Annotations

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("order")
public @interface OrderTx {
}

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("account")
public @interface AccountTx {
}
```

```
public class TransactionalService {

  @OrderTx
  public void setSomething(String name) { ... }

  @AccountTx
  public void doSomething() { ... }
}
```

http://www.flickr.com/photos/vladdythephotogeek/161292150/

Transaction Propagation

Required

REQUIRED
Transaction
Caller
Transactional method 1
Transactional method 2
Transaction created, committed or rolled back as needed
Method 2 executes in the existing transaction.

Requires New

he underlying physical transactions are different and hence can commit or roll back independently

Nested

PROPAGATION_NESTED uses a *single* physical transaction with **multiple savepoints** that it can roll back to.

Such partial rollbacks allow an inner transaction scope to **trigger a rollback *for its scope***, with the outer transaction being able to continue the physical transaction despite some operations having been rolled back. This setting is typically mapped onto JDBC savepoints, so will only work with JDBC resource transactions.

DAO

Support

Exception Translation

Automaticaly when using *@Repository*

JDBC Template Quering

```
int rowCount = this.jdbcTemplate.queryForInt("select count(*) from t_actor");

String lastName = this.jdbcTemplate.queryForObject(
        "select last_name from t_actor where id = ?",
        new Object[]{1212L}, String.class);

Actor actor = this.jdbcTemplate.queryForObject(
        "select first_name, last_name from t_actor where id = ?",
        new Object[]{1212L},
        new RowMapper<Actor>() {
            public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
                Actor actor = new Actor();
                actor.setFirstName(rs.getString("first_name"));
                actor.setLastName(rs.getString("last_name"));
                return actor;
            }
        });

List<Actor> actors = this.jdbcTemplate.query(
        "select first_name, last_name from t_actor",
        new RowMapper<Actor>() {
            public Actor mapRow(ResultSet rs, int rowNum) throws SQLException {
                Actor actor = new Actor();
                actor.setFirstName(rs.getString("first_name"));
                actor.setLastName(rs.getString("last_name"));
                return actor;
            }
        });
```

JDBC Template Updating

```
this.jdbcTemplate.update(
        "delete from actor where id = ?",
        Long.valueOf(actorId));

this.jdbcTemplate.update(
        "update t_actor set = ? where id = ?",
        "Banjo", 5276L);


this.jdbcTemplate.update(
        "insert into t_actor (first_name, last_name) values (?, ?)",
        "Leonor", "Watling");
```

ORM Support

JDO	JPA	Easier Testing	General resource management
Hibernate	iBatis	Common exceptions	Integrated transaction management

http://www.flickr.com/photos/bitzcelt/2903751010/sizes/l/

The Web

The DispatcherServlet

The DispatcherServlet

```
<web-app>

    <servlet>
        <servlet-name>example</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>example</servlet-name>
        <url-pattern>*.form</url-pattern>
    </servlet-mapping>

</web-app>
```

the framework looks for a file named [servlet-name]-servlet.xml in the WEB-INFdirectory

@RequestMapping

```
@Controller
public class ClinicController {

    private final Clinic clinic;

    @Autowired
    public ClinicController(Clinic clinic) {
        this.clinic = clinic;
    }

    @RequestMapping("/")
    public void welcomeHandler() {
    }

    @RequestMapping("/vets")
    public ModelMap vetsHandler() {
        return new ModelMap(this.clinic.getVets());
    }

}
```

URI Templating

http://www.example.com/users/{userid}

http://www.example.com/users/fred

```
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable String ownerId, Model model) {
  Owner owner = ownerService.findOwner(ownerId);
  model.addAttribute("owner", owner);
  return "displayOwner";
}
```

@RequestMapping

```
@Controller @RequestMapping("/appointments")
public class AppointmentsController {

.....

    @RequestMapping(method = RequestMethod.GET)
    public Map<String, Appointment> get() {
        return appointmentBook.getAppointmentsForToday();
    }

    @RequestMapping(value="/{day}", method = RequestMethod.GET)
    public Map<String, Appointment> getForDay(@PathVariable @DateTimeFormat(iso=ISO.DATE) Date day, Model
model) {
        return appointmentBook.getAppointmentsForDay(day);
    }

    @RequestMapping(value="/new", method = RequestMethod.GET)
    public AppointmentForm getNewForm() {
        return new AppointmentForm();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String add(@Valid AppointmentForm appointment, BindingResult result) {
        if (result.hasErrors()) {
            return "appointments/new";
        }
        appointmentBook.addAppointment(appointment);
        return "redirect:/appointments";
    }
```

@PathVariable

```
@RequestMapping(value="/owners/{ownerId}/pets/{petId}", method=RequestMethod.GET)
public String findPet(@PathVariable String ownerId, @PathVariable String petId,
Model model) {
  Owner owner = ownerService.findOwner(ownderId);
  Pet pet = owner.getPet(petId);
  model.addAttribute("pet", pet);
  return "displayPet";
}
```

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

  @RequestMapping("/pets/{petId}")
  public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
    // implementation omitted
  }
}
```

Advanced @RequestMapping

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

  @RequestMapping(value = "/pets/{petId}", params="myParam=myValue")
  public void findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
  }
}
```

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {

@RequestMapping(value = "/pets", method = RequestMethod.POST, headers="content-type=text/*")
  public void addPet(Pet pet, @PathVariable String ownerId) {
    // implementation omitted
  }
}
```

@RequestBody

```
@RequestMapping(value = "/something", method = RequestMethod.PUT)
public void handle(@RequestBody String body, Writer writer) throws
IOException {
  writer.write(body);
}
```

- ByteArrayHttpMessageConverter converts byte arrays.
- StringHttpMessageConverter converts strings.
- FormHttpMessageConverter converts form data to/from a MultiValueMap<String, String>.
- SourceHttpMessageConverter converts to/from a javax.xml.transform.Source.
- MarshallingHttpMessageConverter converts to/from an object using the org.springframework.oxm package.

@ResponseBody

```
@RequestMapping(value = "/something", method = RequestMethod.PUT)
@ResponseBody
public String helloWorld()  {
  return "Hello World"
}
```

As with @RequestBody, Spring converts the returned object to a response body by using an HttpMessageConverter.

Getting JSON from the Server

```
 $(document).ready(function() {
    // check name availability on focus lost
    $('#name').blur(function() {
        checkAvailability();
    });
});

function checkAvailability() {
    $.getJSON("account/availability", { name: $('#name').val() }, function(availability) {
        if (availability.available) {
            fieldValidated("name", { valid : true });
        } else {
            fieldValidated("name", { valid : false,
                message : $('#name').val() + " is not available, try " + availability.suggestions });
        }
    });
}
```

```
@RequestMapping(value="/availability", method=RequestMethod.GET)
public @ResponseBody AvailabilityStatus getAvailability(@RequestParam String name) {
    for (Account a : accounts.values()) {
        if (a.getName().equals(name)) {
            return AvailabilityStatus.notAvailable(name);
        }
    }
    return AvailabilityStatus.available();
}
```

Posting JSON to the Server

```
$("#account").submit(function() {
    var account = $(this).serializeObject();
    $.postJSON("account", account, function(data) {
        $("#assignedId").val(data.id);
        showPopup();
    });
    return false;
});
```

```
@RequestMapping(method=RequestMethod.POST)
public @ResponseBody Map<String, ? extends Object> create(@RequestBody Account
account, HttpServletResponse response) {

    Set<ConstraintViolation<Account>> failures = validator.validate(account);
    if (!failures.isEmpty()) {
        response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
        return validationMessages(failures);
    } else {
        accounts.put(account.assignId(), account);
        return Collections.singletonMap("id", account.getId());
    }

}
```

@ModelAttribute (2 usages)

```
@Controller
@RequestMapping("/owners/{ownerId}/pets/{petId}/edit")
@SessionAttributes("pet")
public class EditPetForm {

    @ModelAttribute("types")
    public Collection<PetType> populatePetTypes() {
        return this.clinic.getPetTypes();
    }

    @RequestMapping(method = RequestMethod.POST)
    public String processSubmit(
            @ModelAttribute("pet") Pet pet,
            BindingResult result, SessionStatus status) {

        new PetValidator().validate(pet, result);
        if (result.hasErrors()) {
            return "petForm";
        }
        else {
            this.clinic.storePet(pet);
            status.setComplete();
            return "redirect:owner.do?ownerId=" + pet.getOwner().getId();
        }
    }
}
```

@SessionAttributes

```
@Controller
@RequestMapping("/editPet.do")
@SessionAttributes("pet")
public class EditPetForm {
    // ...
}
```

@CookieValue

JSESSIONID=415A4AC178C59DACE0B2C9CA727CDD84

```
@RequestMapping("/displayHeaderInfo.do")
public void displayHeaderInfo(@CookieValue("JSESSIONID") String cookie)  {

  //...

}
```

@RequestHeader

```
Host                    localhost:8080
Accept                  text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language         fr,en-gb;q=0.7,en;q=0.3
Accept-Encoding         gzip,deflate
Accept-Charset          ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive              300
```

```
@RequestMapping("/displayHeaderInfo.do")
public void displayHeaderInfo(@RequestHeader("Accept-Encoding") String encoding,
                              @RequestHeader("Keep-Alive") long keepAlive)  {

  //...

}
```

WebDataBinder

All

```
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
    <property name="cacheSeconds" value="0" />
    <property name="webBindingInitializer">
        <bean class="org.springframework.samples.petclinic.web.ClinicBindingInitializer" />
    </property>
</bean>
```

Controller Only

```
@Controller
public class MyFormController {
    @InitBinder
    public void initBinder(WebDataBinder binder) {
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
        dateFormat.setLenient(false);
        binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, false));
    }
}
```

Interceptors

```
<beans>
    <bean id="handlerMapping"
          class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="interceptors">
            <list>
                <ref bean="officeHoursInterceptor"/>
            </list>
        </property>
        <property name="mappings">
            <value>
                /*.form=editAccountFormController
                /*.view=editAccountFormController
            </value>
        </property>
    </bean>

    <bean id="officeHoursInterceptor"
          class="samples.TimeBasedAccessInterceptor">
        <property name="openingTime" value="9"/>
        <property name="closingTime" value="18"/>
    </bean>
<beans>
```

Interceptors

```
public class TimeBasedAccessInterceptor extends HandlerInterceptorAdapter {

    private int openingTime;
    private int closingTime;

    public void setOpeningTime(int openingTime) {
        this.openingTime = openingTime;
    }

    public void setClosingTime(int closingTime) {
        this.closingTime = closingTime;
    }

    public boolean preHandle(
            HttpServletRequest request,
            HttpServletResponse response,
            Object handler) throws Exception {

        Calendar cal = Calendar.getInstance();
        int hour = cal.get(HOUR_OF_DAY);
        if (openingTime <= hour < closingTime) {
            return true;
        } else {
            response.sendRedirect("http://host.com/outsideOfficeHours.html");
            return false;
        }
    }
}
```

Exception Handler

```
@Controller
public class SimpleController {

@ExceptionHandler(IOException.class)
  public String handleIOException(IOException ex, HttpServletRequest request) {
    return ClassUtils.getShortName(ex.getClass());
  }
}
```

E-Tag Support

An ETag (entity tag) is an HTTP response header returned by an HTTP/1.1 compliant web server used to determine change in content at a given URL. It can be considered to be the more sophisticated successor to the Last-Modified header. When a server returns a representation with an ETag header, the client can use this header in subsequent GETs, in an If-None-Match header. If the content has not changed, the server returns 304: Not Modified.

E-Tag Support

Support for ETags is provided by the servlet filter **ShallowEtagHeaderFilter**. It is a plain Servlet Filter, and thus can be used in combination with any web framework. The ShallowEtagHeaderFilter filter creates so-called shallow ETags (as opposed to deep ETags, more about that later). **The filter caches the content of the rendered JSP** (or other content), **generates an MD5 hash over that, and returns that as an ETag header in the response**. The next time a client sends a request for the same resource, it uses that hash as the If-None-Match value. The filter detects this, renders the view again, and **compares the two hashes**. If they are equal, a 304 is returned. This filter will not save processing power, as the view is still rendered. **The only thing it saves is bandwidth, as the rendered response is not sent back over the wire**.

```
<filter>
  <filter-name>etagFilter</filter-name>
    <filter-class>org.springframework.web.filter.ShallowEtagHeaderFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>etagFilter</filter-name>
  <servlet-name>petclinic</servlet-name>
</filter-mapping>
```

REST Template

DELETE - delete(String, String...)
GET - getForObject(String, Class, String...)
HEAD - headForHeaders(String, String...)
OPTIONS - optionsForAllow(String, String...)
POST - postForLocation(String, Object, String...)
PUT - put(String, Object, String...)

```
String result = restTemplate.
   getForObject("http://example.com/hotels/{hotel}/bookings/{booking}",
      String.class, "42", "21");
```

HTTP Method Conversion

HTML Support Just GET and POST Mehtods. Solution JavaScript or Hidden Parameter

Hidden Parameter
hiden **_method** attribute

```
<form:form method="delete">.../>
```

```
@RequestMapping(method = RequestMethod.DELETE)
```

```
<filter>
    <filter-name>httpMethodFilter</filter-name>
    <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>httpMethodFilter</filter-name>
    <servlet-name>petclinic</servlet-name>
</filter-mapping>
```

Views

JSP / JSTL	Freemarker	EXCEL	PDF
Velocity	JSON (Jackson)	XML (OXM)	ATOM
TILES	XSTL	RSS	Jasper Reports

Remoting

RMI

Spring HTTP Invoker
(Serialization)

Hessian
(lightweight binary HTTP-based protocol)

Burlap
(XML-based alternative to Hessian)

JMS

JAX-RPC

JAX-WS

http://www.flickr.com/photos/laffy4k/367822192/sizes/o/

Asynchronous Execution and Scheduling

The Task Namespace (S3+)

Executors are the Java 5 name for the concept of thread pools

```
<task:scheduler id="myScheduler" pool-size="10"/>

<task:executor id="executor" pool-size="10"/>


<task:executor id="executorWithPoolSizeRange"
                   pool-size="5-25"
                   queue-capacity="100"/>

<task:scheduled-tasks scheduler="myScheduler">
    <task:scheduled ref="someObject" method="someMethod" fixed-rate="5000"/>
    <task:scheduled ref="anotherObject" method="anotherMethod" cron="*/5 * * * * MON-FRI"/>
<task:scheduled-tasks/>
```

Annotation Support for Scheduling

```
@Scheduled(fixedDelay=5000)
public void doSomething() {}

@Scheduled(fixedRate=5000)
public void doSomething() {}

@Scheduled(cron="*/5 * * * * MON-FRI")
public void doSomething() {}
```

Annotation Support for Asynchronous Execution

```
@Async
void doSomething() {}

@Async
void doSomething(String s) {}

@Async
Future<String> returnSomething(int i) {}
```

Enabling Annotation-Driven Taks

```
<task:annotation-driven executor="myExecutor" scheduler="myScheduler"/>

<task:executor id="myExecutor" pool-size="5"/>

<task:scheduler id="myScheduler" pool-size="10"/>}
```

OpenSymphony Quartz Integration

```
<bean name="exampleJob" class="org.springframework.scheduling.quartz.JobDetailBean">
  <property name="jobClass" value="example.ExampleJob" />
  <property name="jobDataAsMap">
    <map>
      <entry key="timeout" value="5" />
    </map>
  </property>
</bean>
```

```
public class ExampleJob extends QuartzJobBean {

  private int timeout;
  public void setTimeout(int timeout) {
    this.timeout = timeout;
  }

  protected void executeInternal(JobExecutionContext ctx) throws JobExecutionException {}
}
```

OpenSymphony Quartz Integration

```
 <bean id="jobDetail" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
  <property name="targetObject" ref="exampleBusinessObject" />
  <property name="targetMethod" value="doIt" />
  <property name="concurrent" value="false" />
</bean>

<bean id="simpleTrigger" class="org.springframework.scheduling.quartz.SimpleTriggerBean">
    <property name="jobDetail" ref="jobDetail" />
    <property name="startDelay" value="10000" /> <!-- 10 seconds -->
    <property name="repeatInterval" value="50000" /> <!-- repeat every 50 seconds -->
</bean>

<bean id="cronTrigger" class="org.springframework.scheduling.quartz.CronTriggerBean">
    <property name="jobDetail" ref="exampleJob" />
    <property name="cronExpression" value="0 0 6 * * ?" /> <!-- repeat every 50 seconds -->
</bean>

<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
    <property name="triggers">
        <list>
            <ref bean="cronTrigger" />
            <ref bean="simpleTrigger" />
        </list>
    </property>
</bean>
```

http://www.flickr.com/photos/mr_beaver/3515811628/sizes/l/

Dynamic language support

The Lang Tags

- <lang:jruby/> (JRuby)
- <lang:groovy/> (Groovy)
- <lang:bsh/> (BeanShell)

Groovy Example

Messenger.groovy

```
// from the file 'Messenger.groovy'
package org.springframework.scripting.groovy;
import org.springframework.scripting.Messenger

class GroovyMessenger implements Messenger {
    String message
}
```

Java Interface

Spring Configuration

```
<lang:groovy id="messenger" script-source="classpath:Messenger.groovy">
        <lang:property name="message" value="I Can Do The Frug" />
</lang:groovy>
```

Ruby Example

Messenger.groovy

```
<lang:jruby id="messenger" script-interfaces="org.springframework.scripting.Messenger">
    <lang:inline-script>
require 'java'

include_class 'org.springframework.scripting.Messenger'

class RubyMessenger &lt; Messenger

 def setMessage(message)
  @@message = message
 end

 def getMessage
  @@message
 end

end
    </lang:inline-script>
    <lang:property name="message" value="Hello World!" />
</lang:jruby>
```

Java Interface

What Else?

JMX

JCA

E-Mail

References

- http://blog.springsource.com/
- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/index.html
- http://www.infoq.com/presentations/Whats-New-in-Spring-3.0

THANKS!