Spring 3.0
What‘s new

Massimiliano Dessì, Pronetics S.p.A.

Speaker

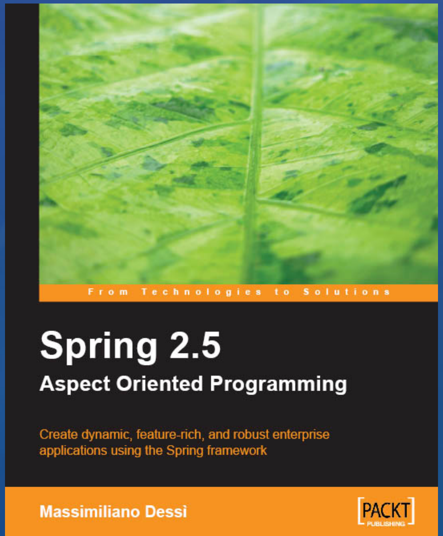
Software Engineer and Architect ProNetics

Founder Spring Italian User Group

Presidente JugSardegna Onlus

Committer/Contributor OpenNMS – MongoDB

Autore Spring 2.5 Aspect Oriented programming


From Technologies to Solutions
Spring 2.5
Aspect Oriented Programming
Create dynamic, feature-rich, and robust enterprise applications using the Spring framework
Massimiliano Dessi
PACKT PUBLISHING

Agenda

- Allineamento codice con Java 5
- Java based bean metadata
- Expression Language
- OXM
- REST
- Declarative Validation
- Embedded Database

Allineamento codice Java 5

Core API allineate con Java 5
recupero di bean senza cast:

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

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

ApplicationContext

```
<?xml version="1.0" encoding="UTF-8"?>

…

 <bean id="uno" class="it.jax.italia.Uno”>

  <constructor-arg index="0” value="pippo"/>

 </bean>


 <bean id="due" class="it.jax.italia.Due" p:nome="Pluto"/>


</beans>
```

Recupero bean

```
ClassPathXmlApplicationContext ctx = new
  ClassPathXmlApplicationContext(”beans.xml");

 Uno one = ctx.getBean("uno", Uno.class);

 Map<String,Due> beans = ctx.getBeansOfType(Due.class);
```

ApplicationEvent

```
public class MyPublisher implements ApplicationEventPublisherAware {

    private ApplicationEventPublisher pub;

    public void setApplicationEventPublisher(
                    ApplicationEventPublisher publisher) {
        pub = publisher;
    }

    public void execute(){
        Evento event = new Evento(this, "Hello from mars !");
        pub.publishEvent(event);
    }
}
```

ApplicationListener

Ora possiamo ricevere gli eventi del tipo che ci interessa anzichè dover utilizzare l'instanceof per discriminare

```
public class EventoListener implements ApplicationListener<Evento> {
  public void onApplicationEvent(Evento event) {
    ..
  }
}
```

Java based bean metadata

Ora è possibile configurare l’applicationContext tramite codice Java

```
@Configuration
public class Config {

   private @Value("#{configProperties.url}") String url;

   @Bean(name="tre")
   public Tre treService() {
       return new Tre(url);
   }

   @Bean(name="quattro")
   public Quattro quattroService(){
       return new Quattro(treService());
   }
}
```

Java based bean metadata

Comunichiamo quale classe contiene la configurazione

e l’eventuale properties file.

```
<?xml version="1.0" encoding="UTF-8"?>
<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"
xmlns:util="http://www.springframework.org/schema/util"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

 <context:component-scan base-package="it.jax.bean.metadata"/>
 <util:properties id="configProperties"
                  location="classpath:it/jax/italia/bean/metadata/config.properties"/>

</beans>
```

SpEL

Spring Expression Language

Si tratta di un linguaggio molto espressivo che supporta l'interrogazione e la modifica di un grafo di oggetti a runtime, utilizzabile nella configurazione xml, nelle annotazioni nelle firme dei metodi, e questo in tutti I prodotti dello Spring Portfolio

SpEL

```
#{T(java.util.Calendar).getInstance()}

#{T(Integer).toString(675)}

#{T(String).CASE_INSENSITIVE_ORDER.compare("d","D")}

#{new java.io.File('/tmp/work').mkdir()}
```

Collections

Proiezioni

```
<collection-name>.![collection-expression]
```

Selezioni

```
<collection-name>.?[collection-expression]
```

Variabili implicite globali

systemProperties

systemEnvironment

Variabili implicite runtime Web

servletContext

contextProperties – Web application init parameters

contextAttributes – attributi Servlet context

Request

Session

servletConfig

Variabili implicite runtime Web JSF

Request/session/application
requestScope
sessionScope
applicationScopecookie
header/headerValues
param/paramValues
initParam
view
facesContext

Operatori

`== != < > <= >= instanceof`

and or !

+ - * / % ^

<expression>? If-true-result : if-false-result

<expression>?: if-null-result (Elvis with default)

SpEL XML

Utilizzabile nella configurazione XML

dove posso anche riferirli tra loro

```
<bean id="numberGuess" class="org.spring.samples.NumberGuess"

      p:randomNumber="#{ T(java.lang.Math).random() * 100.0 }"/>


<bean id="shapeGuess" class="org.spring.samples.ShapeGuess"

      p:initialShapeSeed="#{ numberGuess.randomNumber }"/>
```

SpEL nel codice

```
private MovieFinder movieFinder;
private String defaultLocale;


@Value("#{ systemProperties['user.region'] }")

public void setDefaultLocale(String defaultLocale){

  this.defaultLocale = defaultLocale;

}


@Autowired

public void configure(MovieFinder movieFinder,

       @Value("#{ systemProperties['user.region'] } String defaultLocale){

  this.movieFinder = movieFinder;

  this.defaultLocale = defaultLocale;

}
```

SpEL Security

```
@PreAuthorize("hasRole('ROLE_SUPERVISOR') or " +
"hasRole('ROLE_TELLER') and " +
"(#account.balance + #amount >= -#account.overdraft)" )
public void post(Account account, double amount);
```

Object XML Mapping

In Spring 3.0 è stato creato il package `org.springframework.oxm` dove troviamo

```
public interface Marshaller {
/**
* Marshals the object graph with the given root into
 the provided Result.
*/
void marshal(Object graph, Result result)
     throws XmlMappingException, IOException;
}
```

Object XML Mapping

e l’ unmarshall

```
public interface Unmarshaller {
/**
* Unmarshals the given provided Source into an object
 graph.
*/
Object unmarshal(Source source)
    throws XmlMappingException, IOException;
}
```

Object XML Mapping

Viene fornita l'implementazione con:

JAXB 2

Castor

XMLBeans

JiBX

XStream

Object XML Mapping

Per utilizzare JAXB2 si utilizza nella configurazione 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
xmlns:oxm=http://www.springframework.org/schema/oxm
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/oxm
http://www.springframework.org/schema/oxm/spring-oxm-3.0.xsd">


<oxm:jaxb2-marshaller id="marshaller">
   <oxm:class-to-be-bound name="org.samples.airline.schema.Airport"/>
   <oxm:class-to-be-bound name="org.samples.airline.schema.Flight"/>
</oxm:jaxb2-marshaller>

<oxm:jaxb2-marshaller id="marshaller"
 contextPath="org.springframework.ws.samples.airline.schema"/>
```

Object XML Mapping

Per utilizzare CASTOR si utilizza nella configurazione 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
xmlns:oxm=http://www.springframework.org/schema/oxm
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/oxm
http://www.springframework.org/schema/oxm/spring-oxm-3.0.xsd">

<bean id="castorMarshaller
  class="org.springframework.oxm.castor.CastorMarshaller"

   p:mappingLocation="classpath:mapping.xml" />
```

Object XML Mapping

Per utilizzare XMLBeans si utilizza nella configurazione 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
xmlns:oxm=http://www.springframework.org/schema/oxm
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/oxm
http://www.springframework.org/schema/oxm/spring-oxm-3.0.xsd">

<oxm:xmlbeans-marshaller id="marshaller"/>
```

Object XML Mapping

Per utilizzare **JiBX** si utilizza nella configurazione 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
xmlns:oxm=http://www.springframework.org/schema/oxm
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/oxm
http://www.springframework.org/schema/oxm/spring-oxm-3.0.xsd">

<oxm:jibx-marshaller id="marshaller"
  target-class="org.springframework.ws.samples.airline.schema.Flight"/>
```

Object XML Mapping

Per utilizzare **XSTREAM** si utilizza nella configurazione XML

```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
xmlns:oxm=http://www.springframework.org/schema/oxm
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/oxm
http://www.springframework.org/schema/oxm/spring-oxm-3.0.xsd">

<bean id="xstreamMarshaller"
  class="org.springframework.oxm.xstream.XStreamMarshaller">
  <property name="aliases">
   <props>
       <prop key="Flight">org.springframework.oxm.xstream.Flight</prop>
   </props>
  </property>
</bean>
```

Date Time support

```
public class AnnotationDrivenDateTimeFormattingShowcase {

  // by default, printed & parsed using localized short datetime format (no
  annotation required) e.g. Locale.US=mm/dd/yyyy h:mm:ss a
  private Date dateField;

  // printed & parsed using localized short date format e.g. Locale.US=mm/dd/yyyy
  @DateTimeFormat(dateStyle = Style.SHORT)
  private Date shortDateField;

  // printed & parsed using localized short time format e.g Locale.US=h:mm:ss a
  @DateTimeFormat(timeStyle = Style.SHORT)
  private Date shortTimeField;

  // printed & parsed using custom datetime pattern
  @DateTimeFormat(pattern = "yyyy/mm/dd")
  private Date dateFieldCustomPattern;
```

Date Time support

```
// by default, printed & parsed using localized short datetime format (no
annotation required) e.g. Locale.US=mm/dd/yyyy h:mm:ss a
private DateTime jodaDateField;

// printed & parsed using localized short date format e.g. Locale.US=mm/dd/yyyy
@DateTimeFormat(dateStyle = Style.SHORT)
private DateTime shortJodaDateField;

// printed & parsed using default ISO date time format yyyy-mm-ddThh:mm:ss.SSSZ
@ISODateTimeFormat
private Date isoDateField;

// by default, printed & parsed using localized short datetime format (no
annotaiton required) e.g. Locale.US=mm/dd/yyyy h:mm:ss a
private Calendar calendar;

// printed & parsed using localized short date format e.g. Locale.US=mm/dd/yyyy
@DateTimeFormat(dateStyle = Style.SHORT)
private Calendar shortCalendar;
```

Date Time support

```
// printed & parsed using short datetime format e.g. Locale.US=mm/dd/yyyy h:mm:ss
   @DateTimeFormat(dateStyle = Style.SHORT, timeStyle=Style.SHORT)
   private Long millis;

   /* @Controller method using @DateTimeFormat annotation on method param to
   specify format of 'dateParam' request parameter*/
   @RequestMapping
   public void handlerMethod(@RequestParam @DateTimeFormat(dateStyle =
   Style.SHORT) Date dateParam) {

   }
}
```

MVC namespace

Spring 3.0 RC2

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

  <!-- Turns on support for mapping requests to Spring MVC @Controller methods
       Also registers default Formatters and Validators for use across all
  @Controllers -->

       <mvc:annotated-controllers />

</beans>
```

MVC namespace

`<mvc:annotated-controllers />` indica l'abilitazione dei seguenti bean

```
<bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping"
   p:order=1/>


 <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
   <property name="webBindingInitializer">
      <bean class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">
             <property name="conversionService" ref="conversionService" />
             <property name="validator" ref="validator" />
      </bean>
    </property>
 </bean>

 <bean id="conversionService"
   class="org.springframework.samples.petclinic.util.PetclinicConversionServiceFactory" />

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

REST

- Uri Template support (Client e Controller)
- Rappresentazione HTML, XML, RSS, Atom, PDF, Excel, JSON
- Filtro servlet per il supporto PUT/DELETE
- Costruito su SpringMVC

REST

Possiamo utilizzare delle mappature

Rest style:

```
@RequestMapping(value = "/articles/{id}", method = RequestMethod.GET)
public ModelAndView getArticle(@PathVariable("id") String id) {
      return new ModelAndView("articleView", "article",
                                         articleFacade.getArticle(id));
}
```

E avere content negotiation nel view resolver

Accedere ai cookie e agli header della request attraverso le annotazioni

```
@RequestMapping("/show")
 public ModelAndView getInfo(@RequestHeader("region") long regionId,
   @CookieValue("language") String langId) {
 ...
}
```

REST

Sopratutto abbiamo un RESTTemplate che come glialtriSpringtemplate semplifica le chiamate REST

HTTP	RestTemplate
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...)

REST

Gli oggetti da e per i metodi del RestTemplate passano attraverso messageconverters per essere convertiti.

```
ByteArrayHttpMessageConverter
StringHttpMessageConverter
FormHttpMessageConverter
SourceHttpMessageConverter
MarshallingHttpMessageConverter
BufferedImageHttpMessageConverter
```

Il RestTemplate ci permette quindi di fare chiamate da e verso servizi REST in modo semplificato.

REST

Se abbiamo un controller che ha un mappatura di questo tipo:

```
@Controller
public class RestController {

 @RequestMapping(value= "/books/{book}/{pg}”,method = RequestMethod.GET)
 public String getArticle(@PathVariable("book") String book,
                          @PathVariable("pg") String pg){
  …
}
```

REST

Possiamo interrogare il controller precedente

attraverso il RestTemplate

```
RestTemplate restTemplate = new RestTemplate();
Map<String,String> vars = new HashMap<String,String>();
vars.put("book", "sandokan");
vars.put("pg", "21");
String uri = "http://example.com/books/{book}/{pg}";
String result = restTemplate.getForObject(uri,
                                        String.class, vars);
```

Validazione dichiarativa

Nelle classi

```
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;

public class MyBean {

    @NotNull
    @Max(64)
    private String name;

    @Min(0)
    private int age;
}
```

Validazione dichiarativa

Configuriamo un JSR 303 validator

```
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
   <property name="webBindingInitializer”>
        <bean class="org.springframework.web.bind.support.ConfigurableWebBindingInitializer">
            <property name="validator" ref="validator" />
        </bean>
   </property>
</bean>
```

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

Possiamo così usare l’annotazione `@Valid` nei metodi dei Controller

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

Database embedded

Supporto per H2, Derby, HSQL

Nella configurazione

```
<jdbc:embedded-database id="dataSource">
  <jdbc:script location="classpath:schema.sql"/>
  <jdbc:script location="classpath:test-data.sql"/>
</jdbc:embedded-database>
```

Database embedded

Utile soprattutto per l' utilizzo nei test

```
EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
EmbeddedDatabase db = builder.type(EmbeddedDatabaseType.H2).
           script("schema.sql").script("test-data.sql").build();

// do stuff against the db (EmbeddedDatabase extends
javax.sql.DataSource)

db.shutdown();
```

Domande ?

Grazie per l'attenzione !

Massimiliano Dessì

desmax74 at yahoo.it

massimiliano.dessi at pronetics.it

http://twitter.com/desmax74

http://jroller.com/desmax

http://www.linkedin.com/in/desmax74

http://wiki.java.net/bin/view/People/MassimilianoDessi

http://www.jugsardegna.org/vqwiki/jsp/Wiki?MassimilianoDessi

Spring Framework Italian User Group

http://it.groups.yahoo.com/group/SpringFramework-it