Spring Annotations

Created & Presented by

Sangeeta Joshi

Agenda

- Context Configuration Annotations
- Stereotyping Annotations
- Spring MVC Annotations
- Transaction Annotations
- JSR-250 ANNOTATIONS

Annotations

- Use of annotations requires that certain BeanPostProcessors be registered within Spring container.
- These can be registered as individual bean definitions
- but they can also be implicitly registered by including the following tag in an XML-based Spring configuration (notice the inclusion of the 'context' namespace):

```
<?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:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">
<context:annotation-config/>
```

Context Configuration Annotations

These annotations are used by Spring to guide creation and injection of beans.

Annotation	Use	Description
@Autowired	Constructor, Field, Method	Declares a constructor, field, setter method, or configuration method to be autowired by type. Items annotated with @Autowired do not have to be public.
@Configurable	Type	Used with to declare types whose properties should be injected, even if they are not instantiated by Spring. Typically used to inject the properties of domain objects.
@Order	Type, Method, Field	Defines ordering, as an alternative to implementing the org. springframework.core.Ordered interface.
@Qualifier	Field, Parameter, Type, Annotation Type	Guides autowiring to be performed by means other than by type.
@Required	Method (setters)	Specifies that a particular property must be injected or else the configuration will fail.
@Scope	Type	Specifies the scope of a bean, either singleton, prototype, request, session, or some custom scope.

Annotations

@Required

Default behavior is to treat annotated methods, constructors, and fields as indicating **required dependencies**.

- This annotation indicates that the affected bean property must be populated at configuration time: either through an explicit property value in a bean definition or through autowiring.
- The container will throw an exception if the affected bean property has not been populated;
- this allows for eager and explicit failure, avoidingNullPointerExceptions or the like later on.

@Required

Ensuring That Required Properties are Set

- To ensure that a property is injected with a value, use the @Required annotation:

- ```
  @Required
  public void setTreasureMap(TreasureMap treasureMap)
        {
              this.treasureMap = treasureMap;
         }
  ```
  ```

Annotations

@Resource

- Spring supports injection using @Resource annotation on fields or bean property setter methods.
- @Resource takes a 'name' attribute, and by default Spring will interpret that value as the bean name to be injected. In other words, it follows *by-name* semantics

Annotations

@Autowired

- @Autowired annotation may be applied to "traditional" setter methods
- The annotation may be applied to methods with arbitrary names and multiple arguments.
- @Autowired annotation may even be applied on constructors and fields:

Annotations

@Qualifier

- Since autowiring by type may lead to multiple candidates, it is often necessary to have more control over the selection process.
- One way to accomplish this is with Spring's @Qualifier annotation.
- This allows for associating qualifier values with specific arguments, narrowing the set of type matches so that a specific bean is chosen for each argument.
- In the simplest case, this can be a plain descriptive value

Autowiring

Autowiring Bean Properties

- A typical Spring bean might have its properties wired something like this:

- <bean id="pirate" class="Pirate"> <constructor-arg value="Long John Silver" /> <property name="treasureMap" ref="treasureMap" /> </bean>

Autowiring

Autowiring Bean Properties

```
public class Pirate
 {
 private String name;
private TreasureMap treasureMap;
public Pirate(String name)
{ this.name = name; }
@Autowired
public void setTreasureMap(TreasureMap treasureMap)
{
 this.treasureMap = treasureMap;
}
 }
```

Autowiring

Autowiring Bean Properties

A typical Spring bean might have its properties wired something like this:

```
<bean id="pirate" class="Pirate">

    <constructor-arg value="Long John Silver" />

     <property name="treasureMap" ref="treasureMap" />

</bean>
```

Autowiring Without Setter Methods

- @Autowired

```
public void directionsToTreasure(TreasureMap treasureMap)
 {
      this.treasureMap = treasureMap;
 }
```

- ***And even on member variables:***

```
@Autowired
 private TreasureMap treasureMap;
```

- ***And To resolve any autowiring ambiguity, use @Qualifier***

```
@Autowired
@Qualifier("mapToTortuga")
 private TreasureMap treasureMap;
```

JSR-250 ANNOTATIONS

@PostConstruct and @PreDestroy methods, you can declare methods that hook into a bean's lifecycle.
For example, consider the following methods added to the Pirate class:

```
public class Pirate
{ ...
    @PostConstruct
        public void wakeUp()
        { System.out.println("Yo ho!");
        }

    @PreDestroy
        public void goAway()
       { System.out.println("Yar!");
        }
}
```

Stereotyping Annotations

- These annotations are used to stereotype classes with regard to the application tier that they belong to.
- Classes that are annotated with one of these annotations will automatically be registered in the Spring application context if ***<context:component-scan>*** is in the Spring XML configuration.
- In addition, if a PersistenceExceptionTranslationPostProcessor is configured in Spring, any bean annotated with @Repository will have SQLExceptions thrown from its methods translated into one of Spring's unchecked DataAccessExceptions.

Stereotyping Annotations

Annotation	Use	Description
@Component	Type	Generic stereotype annotation for any Spring-managed component.
@Controller	Type	Stereotypes a component as a Spring MVC controller.
@Repository	Type	Stereotypes a component as a repository. Also indicates that SQLExceptions thrown from the component's methods should be translated into Spring DataAccessExceptions.
@Service	Type	Stereotypes a component as a service.

Spring MVC Annotations

Spring MVC Annotations

These annotations were introduced in Spring 2.5

- to make it easier to create Spring MVC applications with minimal XML configuration

- and without extending one of the many implementations of the Controller interface.

Spring MVC Annotations

Annotation	Use	Description
@Controller	Type	Stereotypes a component as a Spring MVC controller.
@InitBinder	Method	Annotates a method that customizes data binding.
@ModelAttribute	Parameter, Method	When applied to a method, used to preload the model with the value returned from the method. When applied to a parameter, binds a model attribute to the parameter. table
@RequestMapping	Method, Type	Maps a URL pattern and/or HTTP method to a method or controller type.
@RequestParam	Parameter	Binds a request parameter to a method parameter.
@SessionAttributes	Type	Specifies that a model attribute should be stored in the session.

Spring MVC Annotations

- Before we can use annotations on Spring MVC controllers, we'll need to add a few lines of XML to tell Spring that our controllers will be annotation-driven.

- First, we won't have to register each of our controllers individually as <bean>s, we'll need a <context:component-scan>:

  <context:component-scan
  base-package="com.habuma.pirates.mvc"/>

- In addition to auto registering @Component-annotated beans, <context:component-scan> also auto registers beans that are annotated with @Controller.

- we'll also need to tell Spring to honor the other Spring MVC annotations. For that : ***<context:annotation-config> :*** ***<context:annotation-config/>***

Creating a Simple MVC Controller

```
@Controller

@RequestMapping("/home.htm")

    public class HomePage
    {
        @RequestMapping(method = RequestMethod.GET)

          public String showHomePage(Map model)
          {
              List<Pirate> pirates = pirateService. getPirateList();
               model.add("pirateList", pirates); return "home";
            }

         @Autowired
           PirateService pirateService;

      }
```

Creating a Form-Handling Controller

- In a pre-2.5 Spring MVC application, form-processing controllers would typically extend SimpleFormController (or some similar base class).

- But with Spring 2.5, a form-processing controller just has a method that is annotated to handle the HTTP POST request:

Creating a Form-Handling Controller

```
@Controller
@RequestMapping("/addPirate.htm")
public class AddPirateFormController
 {
    @RequestMapping(method = RequestMethod.GET)
        public String setupForm(ModelMap model)
      {          return "addPirate";       }
    @ModelAttribute("pirate")
       public Pirate setupPirate()
        { Pirate pirate = new Pirate();
              return pirate; }
    @RequestMapping(method = RequestMethod.POST)
           protected String addPirate(
             @ModelAttribute("pirate")
                    Pirate pirate)
         { pirateService.addPirate(pirate); return "pirateAdded"; }
       @Autowired PirateService pirateService;
  }
```

Annotating Transactional Boundaries

- To use Spring's support for annotation-declared transactions, you'll first need to add a small amount of XML to the Spring configuration:

```
<?xml version="1.0" encoding="UTF-8"?> <beans
xmlns="http://www.springframework.org/schema/ beans"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/ schema/beans
http://www.springframework.org/schema/beans/ springbeans-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx- 2.5.xsd">
<tx:annotation-driven /> ... </beans>
```

Transaction Annotations

- The @Transactional annotation is used along with the <tx:annotation-driven> element to declare transactional boundaries and rules as class and method metadata in Java.

Annotation	Use	Description
@Transactional	Method, Type	Declares transactional boundaries and rules on a bean and/or its methods.

Annotating Transactional Boundaries

- The <tx:annotation-driven> element tells Spring to keep an eye out for beans that are annotated with @Transactional.
- In addition, you also need a platform transaction manager bean declared in the Spring context.
- For example, if your application uses Hibernate, you'll want to include the HibernateTransactionManager:

```
<bean id="transactionManager"
    class="org.springframework.orm.hibernate3.
     HibernateTransactionManager">
<property name="sessionFactory"
       ref="sessionFactory" />
 </bean>
```

Annotating Transactional Boundaries

```
@Transactional (propagation=Propagation.SUPPORTS,
readOnly=true)
public class TreasureRepositoryImpl implements TreasureRepository
 { ...
      @Transactional(propagation=Propagation.REQUIRED,
            readOnly=false)
          public void storeTreasure(Treasure treasure)

          {
            ..

            }


 ... }
```

Annotating Transactional Boundaries

- At the class level, @Transactional is declaring that all methods should support transactions and be read-only.
- But, at the method-level, @Transactional declares that the storeTreasure() method requires a transaction and is not read-only.
- Note that for transactions to be applied to @Transactionalannotated classes, those classes must be wired as beans in Spring.

JSR-250 ANNOTATIONS

- Spring also supports a few of the annotations defined by JSR-250, which is the basis for the annotations used in EJB 3

Any Questions?