Spring Features

Monday, March 31, 2014 11:23

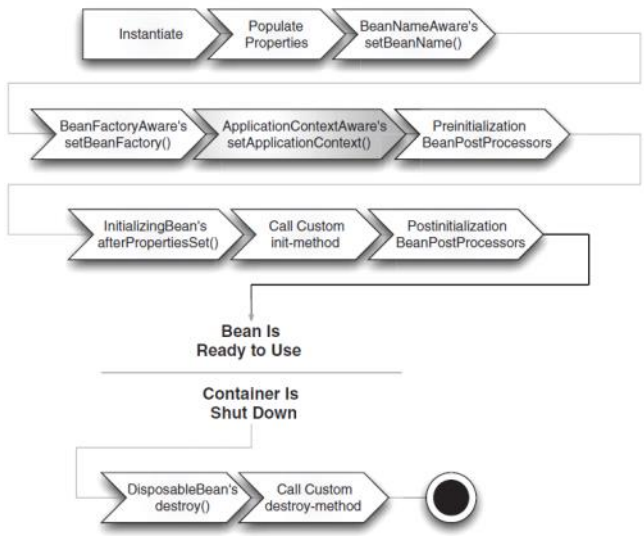
1. **Spring Core Features:**
   a. DI (Dependency injection) or IOC (Inverse of Control)
   b. AOP (aspect-oriented programming)

2. **Bean Factory**
   a. BeanFactory
   b. ApplicationContext

Source Code

SpringLearning

3. **Bean Life cycle**



Step	Description
1. Instantiate	Spring instantiates the bean.
2. Populate properties.	Spring injects the bean's properties.
3. Set bean name.	If the bean implements BeanNameAware, Spring passes the bean's ID to setBeanName().
4. Set bean factory.	If the bean implements BeanFactoryAware, Spring passes the bean factory to setBeanFactory().
4.5. Set application context	if the bean implements the ApplicationContextAware interface, the setApplicationContext() method is called.
5. Postprocess (before initialization).	If there are any BeanPostProcessors, Spring calls their postProcessBeforeInitialization() method.
6. Initialize beans.	If the bean implements InitializingBean, its afterPropertiesSet() method will be called. If the bean has a custom init method declared, the specified initialization method will be called.
7. Postprocess (after initialization).	If there are any BeanPostProcessors, Spring calls their postProcessAfterInitialization() method.
8. Bean is ready to use.	At this point the bean is ready to be used by the application and will remain in the bean factory until it is no longer needed.
9. Destroy bean.	If the bean implements DisposableBean, its destroy() method will be called. If the bean has a custom destroy-method declared, the specified method will be called.

4. **Bean Creation**
   a. Wiring properties
   b. Wiring through constructor

5. **Bean value injection**
   a. Injecting simple values
   b. Referencing other beans
   c. Wiring collections
   d. Wiring Null

6. **Auto wiring**
   a. By name - Attempts to find a bean in the container whose name (or ID) is the same as the name of the property being wired. If a matching bean is not found, the property will remain unwired.
   b. By Type - Attempts to find a single bean in the container whose type matches the type of the property being wired. If no matching bean is found, the property will not be wired. If more than one bean matches, an org.springframework.beans.factory.UnsatisfiedDependencyException will be thrown.
   c. Constructor - Tries to match up one or more beans in the container with the parameters of one of the constructors of the bean being wired. In the event of ambiguous beans or ambiguous constructors, an org.springframework.beans.factory.UnsatisfiedDependencyException will be thrown. **Autowiring by constructor shares the same limitations as byType.**
   d. Autodetect - Attempts to autowire by constructor first and then using byType. Ambiguity is handled the same way as with constructor and byType wiring.

7. **Bean scoping**

Scope	What it does
`singleton`	Scopes the bean definition to a single instance per Spring container (default).
`prototype`	Allows a bean to be instantiated any number of times (once per use).
`request`	Scopes a bean definition to an HTTP request. Only valid when used with a web-capable Spring context (such as with Spring MVC).
`session`	Scopes a bean definition to an HTTP session. Only valid when used with a web-capable Spring context (such as with Spring MVC).
`global-session`	Scopes a bean definition to a global HTTP session. Only valid when used in a portlet context.

Scoping is new to Spring 2.0. Prior to Spring 2.0, you would set a <bean>'s singleton attribute to false to make it a prototype bean.

8. **Controlling bean creation**
   a. Create bean using factory method
      Example: <bean id="<bean name>" class="<class>" factory-method="getInstance"/>
   b. Initializing and destroying beans
      Example: <bean id="<bean name>" class="<class>" init-method="customInit" destroy-method="customDestroy" />
   c. Defaulting init-method and destroy-method
      Example: <**beans** … default-init-method="tuneInstrument" default-destroy-method="cleanInstrument" />
   d. InitializingBean and DisposableBean
      implement two special Spring interfaces: **afterPropertiesSet** and **destroy**.

9. **Declaring parent and child beans**
   To accommodate sub-beaning, the <bean> element provides two special attributes:
   ■ parent—Indicates the id of a <bean> that will be the parent of the <bean> with the parent attribute. The parent attribute is to <bean> what extends is to a Java class.

■ abstract—If set to true, indicates that the <bean> declaration is abstract. That is, it should never be instantiated by Spring.

10. **Bean Name aware, Bean factory ware and Application context aware**
    a. Bean name aware: implement BeanNameAware and override setBeanName(Stirng name)
    b. Bean factory aware: implement BeanFactoryAware and override setBeanFactory(BeanFactory beanFactory)
    c. Application context aware: implement ApplicationContextAware and override setApplicationContext(ApplicationContext applicationContext)

11. **Method injection**
    Example:

```
public class ReturnValueReplacer implements MethodReplacer {

    public Object reimplement(Object obj, Method method, Object[] args) throws Throwable {
        return obj.getClass().getName() + " (" + this.hashCode() + ") (Method replaced)";
    }

}
```