Customizing the nature of a bean

Summary

Description

Lifecycle callbacks

Spring Framework provides various callback interfaces which can change the behavior of bean in the container.

Initialization callbacks

`Upon implementing org.springframework.beans.factory.InitializingBean` interface, set up all properties required for bean and carry out unitization. `InitializingBean` interface specifies the following method.

```
void afterPropertiesSet() throws Exception;
```

In general, the use of `InitializingBean` interface is not recommended, as the code is unnecessarily coupled with String. As an alternative, bean definition can specify the initial method. In a XML base setting, `'init-method'` attribute is used.

```
<bean id="exampleInitBean" class="examples.ExampleBean" init-method="init"/>
public class ExampleBean {

    public void init() {
        // do some initialization work
    }
}
```

The above sample is the same with the sample below.

```
<bean id="exampleInitBean" class="examples.AnotherExampleBean"/>
public class AnotherExampleBean implements InitializingBean {

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

Destruction callbacks

`After implementing org.springframework.beans.factory.DisposableBean` interface, bean can receive callbacks when the container is destroyed. `DisposableBean` interface specifies the following method.

```
void destroy() throws Exception;
```

In general, the use of `DisposableBean` interface is not recommended, as the code is unnecessarily coupled with String. As an alternative, bean definition can specify the initial method. In a XML base setting, `'destroy-method'` attribute is used.

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

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

The above sample is the same with the sample below.

```
<bean id="exampleInitBean" class="examples.AnotherExampleBean"/>
public class AnotherExampleBean implements DisposableBean {

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

Default initialization & destroy methods

Spring container can specify initialization and destruction method to all beans under the same name.

```
public class DefaultBlogService implements BlogService {

    private BlogDao blogDao;

    public void setBlogDao(BlogDao blogDao) {
        this.blogDao = blogDao;
    }

    // this is (unsurprisingly) the initialization callback method
    public void init() {
        if (this.blogDao == null) {
            throw new IllegalStateException("The [blogDao] property must be set.");
        }
    }
}
<beans default-init-method="init">

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

</beans>
```

You can use <beans/> element's 'default-init-method' attribute to specify the default initialization callback method. For the destruction callback method, you can 'default-destroy-method' attribute.

If 'init-method' and 'destroy-method' attributes are defined in <bean/> element, the default value is ignore.

Merging the Life Cycle Mechanisms

In Spring 2.5, there are three types of life cycle mechanisms: InitialzingBean, DisposableBean interface; custom init() and destroy() method; and @PostConstruct and @PreDestroy annotations

If different life cycle mechanisms are used together, the developer should know the sequence of using them. The sequence of initialization is as follows.

1. Method with @PostConstruct annotation
2. afterPropertiesSet() defined in InitializingBean callback interface
3. Custom init() method

The sequence of calling the destruction method is as follows.

1. Method with @PreDestroy annotation
2. destroy() defined in DisposableBean callback interface

3. Custom destroy() mothod

Knowing who you are

BeanFactoryAware

The class that implemented org.springframework.beans.factory.BeanFactoryAware interface can refer to BeanFactory that created itself.

```
public interface BeanFactoryAware {

    void setBeanFactory(BeanFactory beanFactory) throws BeansException;
}
```

Using BeanFactoryWare interface allows informing of the BeanFactory that created itself and searching for other beans, but you would be better to avoid it because it creates coupling with Spring and does not fit the Inversion of Control style.

As an alternative, you can use org.springframework.beans.factory.config.ObjectFactoryCreatingFactoryBean.

The ObjectFactoryCreatingFactoryBean is an implement of FactoryBean and can make reference to the object that finds out bean. The ObjectFactoryCreatingFactoryBean implments BeanFactoryAware interface by itself.

```
package x.y;

public class NewsFeed {

    private String news;

    public void setNews(String news) {
        this.news = news;
    }

    public String getNews() {
        return this.toString() + ": '" + news + "'";
    }
}
package x.y;

import org.springframework.beans.factory.ObjectFactory;

public class NewsFeedManager {

    private ObjectFactory factory;

    public void setFactory(ObjectFactory factory) {
        this.factory = factory;
    }

    public void printNews() {
        // here is where the lookup is performed; note that there is no
        // need to hard code the name of the bean that is being looked up...
        NewsFeed news = (NewsFeed) factory.getObject();
        System.out.println(news.getNews());
    }
}
```

The following setting is an example of using ObjectFactoryCreatingFactoryBean to connect the two classes.

```
<beans>
    <bean id="newsFeedManager" class="x.y.NewsFeedManager">
        <property name="factory">
            <bean
class="org.springframework.beans.factory.config.ObjectFactoryCreatingFactoryBean">
                <property name="targetBeanName">
                    <idref local="newsFeed" />
                </property>
            </bean>
        </property>
    </bean>
    <bean id="newsFeed" class="x.y.NewsFeed" scope="prototype">
        <property name="news" value="... that's fit to print!" />
    </bean>
</beans>
```

BeanNameAware

The bean that implments the org.springframework.beans.factory.BeanNameAware interface can access the ***name*** in container.

References

- Spring Framework - Reference Document / 3.5. Customizing the nature of a bean