SAP Customer Experience

Spring Essentials

We will learn about:

- Spring Essentials Overview
- Scopes for Beans
- Spring Expression Language

Spring Overview

Spring - Overview

- The **Spring Framework** was created as a response to the high complexity of J2EE* specifications and the applications developed under their rules.
- Spring helps developers by removing or lowering the amount of boilerplate code usually required to tie application modules together, thereby simplifying projects and allowing developers to focus on architecture and application logic.
- One of the core elements of Spring is the Inversion of Control Container (IoC Container). The IoC Container provides developers with an Application Context that will:
  - Contain declarations of Java objects (beans) used by the application
  - Control the lifecycle of these beans
  - Handle each Bean's access and reference to other beans it needs – this is called Dependency Injection (see next slide)
  - Handle the application's access to other Spring Framework features

* J2EE – Java 2 Enterprise Edition was an early precursor to Java EE and Jakarta EE

Spring - Dependency Injection

- The **Spring Framework** creates instances from any Java class in the classpath, and it's able to manage them throughout their lifecycle (creation, initialization, destruction, events, etc.)
- These instances are called **Spring beans** (or just **beans**), and they are usuallymajor **components** at the architecture or design levels (DAOs, services, controllers, resources, helper classes, etc.)
- The purpose of this approach is to **decouple** a component's instantiation and configuration from its use. This technique is called **Dependency Injection** (part of the **Inversion of Control** paradigm)
- Spring manages components by:
  - parsing XML files **(Spring XML Config**, the approach most commonly used in SAP Commerce)
  - responding to Spring annotations in Java classes (**Spring Annotation Config**)
  - executing code (**Spring Java Config**)

  or by combining the three above mechanisms

Spring - Bean creation and configuration in XML

- Beans can be defined using **Spring XML Configuration**.
- Create an instance of the bean class using its **default constructor**. The **id** or **name** you provide will be used to reference this component afterwards.

```
<bean id="beanName" class="QualifiedClassName"/>
```

```
<bean id="defaultEmailService"
      class="my.messenger.core.services.impl.DefaultEmailService"/>
```

- The bean can be **configured** by setting any number of its **properties**:

```
<bean id="beanName" class="QualifiedClassName" >
     <property name="propertyNameA" value="literalValue"/>
     <property name="propertyNameB" ref="componentReference"/>
</bean>
```

```
<bean id="defaultEmailService”
      class="my.messenger.core.services.impl.DefaultEmailService”>
   <property name="sendEmails" value="false”/>
   <property name="customerDao" ref="customerDao"/>
</bean>
```

Spring - Accessing one spring bean from another

- In any **Spring-managed component**, provide a public **setter method** to set the property reference

```
public class MyCustomerAccountService ... {

   private DefaultGenericDao<CustomerModel> customerDao;

   public void setCustomerDao(DefaultGenericDao<CustomerModel> customerDao) {
      this.customerDao = customerDao;
   }
   ...
```

- Another way to resolve this reference is to use **Spring Annotations** (@Autowired, @Qualifier, @Resource)
  - Use @Autowired to locate a Spring bean instance of the given type. You get an error if there are multiple instances.

```
@Autowired
private DefaultGμenericDao<CustomerModel> customerDao;
```

- Even better, use **JSR 250 Annotations** (@Resource)

```
@Resource{name="customerDao"}
private DefaultGenericDao<CustomerModel> customerDao;

@Resource
private DefaultGenericDao<CustomerModel> customerDao;
```

Link variable to Spring bean with id "customerDao".
If you omit the name parameter, Spring uses the name of the member variable; therefore, both these lines do the same thing.

Spring - Accessing spring beans from within code

- Anywhere in your code, use a reference to the **Spring ApplicationContext** object to get access to any managed component by id, name, type, or both:

```
private ApplicationContext applicationContext;
applicationContext.getBean("name");
applicationContext.getBean("name",Type);
applicationContext.getBean(Type);
```

- Alternatively, use the convenient **SAP Commerce API**

```
Registry.getApplicationContext().getBean(...);
```

Scopes for Beans

Spring - Scopes for beans

- Spring **default** behavior is to reuse the same component instance wherever it is referenced (**Singleton Components**), but this can be changed by specifying the **scope** attribute in the bean XML element. Use the **Prototype** scope to get a new component instance each time it is referenced.
  - You may also use the **request** and **session** bean scopes in a web app.
- Samples:

```
<bean id="customerEmailContext"
      class="my.training.facades.process.email.context.CustomerEmailContext"
      scope="prototype" />

<bean id="referenceData”
      class="de.hybris.platform.commerceservices.product.data.ReferenceData"
      scope="prototype" />

<bean id="CatalogPerspective”
      class="my.training.cockpits.cmscockpit.session.impl.DefaultWCMSPerspective"
      scope="session" />

<bean id="defaultRequestContextData"
      class="de.hybris.platform.acceleratorservices.data.RequestContextData"
      scope="request" />
```

Spring - Dependency injection via constructor methods

- Spring can configure a bean using any constructor instead of the default one

```
<bean ... >
   <constructor-arg value="literalValue" ref="componentReference" type="propertyType"
                    index="position" name="propertyName" />
</bean>
```

Example

```
<bean id="productFrontendUrlAntPathPattern" class="java.lang.String">
   <constructor-arg>
      <value><![CDATA[/**/p/{code}]]></value>
   </constructor-arg>
</bean>

<bean id="storefrontTenantDefaultFilterChain"
      class="de.hybris.platform.servicelayer.web.PlatformFilterChain">
   <constructor-arg>
      <ref bean="storefrontTenantDefaultFilterChainList"/>
   </constructor-arg>
</bean>
```

- After dealing with the specified constructor method for creating the bean instance, other properties can be configured as usual using the property setter

Spring - Setting other types of properties

- Spring can configure a bean with **any type** of properties; it is not limited to literals or references to other components
- Syntax for setting properties of **java.util.List** type

```
<bean id="acceleratorProductPrimaryImagePopulator" ...>
   <property name="imageFormats">
      <list>
         <value>zoom</value>
         <value>product</value>
         ...
      </list>
   </property>
</bean>

<bean id="defaultFraudService" ...>
   <property name="providers">
      <list>
         <ref bean="internalFraudServiceProvider"/>
         <ref bean="commercialFraudServiceProvider"/>
      </list>
   </property>
</bean>
```

Spring - Setting other types of properties

- Syntax for setting properties of **java.util.Map** type

```
<bean id="acceleratorImageFormatMapping" ...>
   <property name="mapping">
      <map>
         <entry key="superZoom" value="1200Wx1200H"/>
         <entry key="zoom" value="515Wx515H"/>
         ...
      </map>
   </property>
</bean>

<bean id="accBrowserFilterFactory" ...>
   <property name="browserFilters">
      <map>
         <entry key="AbstractPage">
            <list>
               <ref bean="desktopUiExperienceBrowserFilter" />
               <ref bean="mobileUiExperienceBrowserFilter" />
            </list>
         </entry>
      </map>
   </property>
</bean>
```

Spring - Reusing configuration from other beans

- Spring can configure a bean using properties from an already-configured component through a bean **parent** relationship:
- Sample parent bean:

```
<bean id="defaultCustomerAccountService"
      class="de.hybris.platform.commerceservices.customer.impl.DefaultCustomerAccountService">
   <property name="userService" ref="userService"/>
   <property name="modelService" ref="modelService"/>
   <property name="flexibleSearchService" ref="flexibleSearchService"/>
   <property name="i18nService" ref="i18nService"/>
    ...
</bean>
```

- Sample child bean, which inherits all the parent's tag values, which it can change or add to:

  Note: If you specify a `parent` tag and omit a `class` tag, the parent's class will be used for instantiation

```
<bean id="defaultTrainingCustomerAccountService"
      class="my.training.core.services.impl.DefaultBookstoreCustomerAccountService"
      parent="defaultCustomerAccountService">
   <property name="customerDao" ref="customerDao"/>
</bean>
```

Spring - Reusing configuration from other beans

- When using parent beans, the default Spring behaviour is **override** any parent property with the child's configuration. For collection-type properties, it is possible to preserve both groups of values (the parent's and the child's) by using the **merge** attribute:

```
<bean id="accSynchronizationService"
      class="de.hybris.platform.cockpit.services.sync.impl.SynchronizationServiceImpl"
      parent="defaultSynchronizationService">
   <property name="relatedReferencesTypesMap">
      <map merge="true">
         <entry key="Product">
            <list>
               <value>Product.productImages</value>
            </list>
         </entry>
         <entry key="MediaContainer">
            <list>
               <value>MediaContainer.medias</value>
            </list>
         </entry>
      </map>
   </property>
</bean>
```

Spring Expression Language

Spring - SpEL Spring Expression Language

- The Spring Expression Language (**SpEL** for short) is a powerful expression language that supports querying and manipulating a component graph at runtime. The language syntax is similar to Unified EL (JSP or JSF), but offers additional features, most notably method invocation and static member access.

```
<bean id="messageSource"
      class="org.springframework.context...ReloadableResourceBundleMessageSource">
   <property name="defaultEncoding" value="UTF-8"/>
   <property name="cacheSeconds"

value="#{configurationService.configuration.getProperty('store.resourceBundle.cacheSeconds')}"/>
</bean>

<bean id="defaultPaymentDetailsForm"
      class="de.hybris.platform...web.payment.forms.PaymentDetailsForm">
   <property name="cardTypeCode" value="001"/>
   <property name="cardNumber" value="4111111111111111"/>
   <property name="startYear"
             value=" #{T(java.util.Calendar).getInstance().get(T(java.util.Calendar).YEAR) - 1}"/>
   <property name="expiryYear"
             value=" #{T(java.util.Calendar).getInstance().get(T(java.util.Calendar).YEAR) + 1}"/>
</bean>
```

Spring - SpEL Spring Expression Language

- The expression language supports the following functionality
  - Literal expressions
  - Boolean and relational operators
  - Regular expressions
  - Class expressions
  - Accessing properties, arrays, lists, maps
  - Method invocation
  - Relational operators
  - Assignment
  - Calling constructors
  - Bean references
  - Array construction
  - Inline lists
  - Inline maps
  - Ternary operator
  - Variables
  - User defined functions
  - Collection projection
  - Collection selection
  - Templated expressions

Spring - Links

- For more information on the Spring Framework:
  - Spring Framework in SAP Commerce Cloud: https://help.sap.com/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/aa417173fe4a4ba5a473c93eb730a417/8c63621986691014a7e0a18695d7d410.html
  - Spring Usage: https://help.sap.com/docs/SAP_COMMERCE_CLOUD_PUBLIC_CLOUD/7e47d40a176d48ba914b50957d003804/8aef69e986691014a9179a7d4ffc1359.html
  - Spring Framework Reference: http://docs.spring.io/spring/docs/current/spring-framework-reference/html
  - Spring Expression Language: http://docs.spring.io/spring/docs/current/spring-framework-reference/html/expressions.html

Thank you.