Seam Spring

by Marius Bogoevici

1. Seam Spring - Introduction 1
1.1. Features 1
2. Seam Spring - Installation 3
2.1. Maven dependency configuration 3
3. Seam Spring - Architecture and Usage 5
3.1. Accessing Spring artifacts from CDI 5
3.1.1. Accessing Spring application contexts 5
3.1.2. Exposing Spring beans as CDI beans 9
3.2. Importing CDI beans into Spring applications 9
3.2.1. Registering a BeanManager 9
3.2.2. Importing a CDI bean as a Spring bean 10

Seam Spring - Introduction

The Seam Spring module aims to provide a mechanism for integrating the *Spring* [http://www.springframework.org] development model with *CDI* [http://jcp.org/en/jsr/detail?id=299].

1.1. Features

The current version of the module provides support for:

- bootstrapping a Spring application context and making it accessible as a CDI bean;
- registering an independently bootstrapped application context as a CDI bean;
- making Spring beans accessible via CDI (i.e. as managed beans) - for injection and lookup;
- accessing CDI beans from within a Spring application context;

Seam Spring - Installation

2.1. Maven dependency configuration

If you are using *Maven* [http://maven.apache.org/] as your build tool, you can add the following single dependency to your pom.xml file to include the Seam Spring module.

```
<dependency>
  <groupId>org.jboss.seam.spring</groupId>
  <artifactId>seam-spring-core</artifactId>
  <version>${seam.spring.version}</version>
</dependency>
```

Tip

Substitute the expression ${seam.spring.version} with the most recent or appropriate version of Seam Spring.

Seam Spring - Architecture and Usage

The functionality of the Seam Spring module is provided by two sets of components:

- A CDI portable extension for accessing Spring application contexts and managing Spring beans;
- A FactoryBean and corresponding namespace for accessing BeanManagers and importing CDI beans into Spring.

3.1. Accessing Spring artifacts from CDI

The Seam Spring module uses the resource producer pattern for accessing Spring contexts and beans from within CDI. The Spring extension is responsible for producing the actual instances. This mechanism allows the Spring beans to participate in regular CDI injection and the injection targets to be agnostic of the provenience of the injected references, enforcing true separation of concerns. Through this mechanisms Spring contexts can be injected as Spring beans too, if necessary.

The resource producer pattern is used for:

- producing Spring application context instances;
- producing Spring beans;

The registration of Spring application contexts as CDI beans is a prerequisite for accessing the Spring beans that are created by them.

3.1.1. Accessing Spring application contexts

The Seam Spring module can access two types of contexts:

- contexts created by the application (e.g. bootstrapped by Spring's `ContextLoaderListener`); and
- contexts bootstrapped by the extension itself.

3.1.1.1. The `@SpringContext` qualifier

As a general rule, Spring `ApplicationContext` instances that the extension is interacting with are installed as CDI beans with a `@SpringContext` qualifier, with the following structure:

```
@Qualifier
@Inherited
@Documented
@Retention(RetentionPolicy.RUNTIME)
```

```
@Target({ElementType.FIELD, ElementType.METHOD, ElementType.PARAMETER})
public @interface SpringContext {
  String name() default "default";
}
```

The name attribute of the context helps identifying between different ApplicationContexts, if the extension needs to deal with multiple such instances.

Table 3.1. Attributes of @org.jboss.seam.spring.context.SpringContext

Attribute	Type	Significance
`name`	`String`	Unique identifier for a Spring application context bean

3.1.1.2. Producing Spring contexts

CDI applications can install Spring contexts as CDI beans by defining producer fields with the following general pattern:

```
@Produces
@SpringContext
@<Context-Type>
ApplicationContext context;
```

This will create a CDI bean of the ApplicationContext type. The nature of the context (bootstrapped by the extension, or looked up elsewhere) is controlled by a specific annotation. The supported annotations are detailed in the following subsections.

Tip

As a reminder, if the `name` attribute of the `@SpringContext` qualifier is not set, it will be set to 'default'.

3.1.1.2.1. Installing a web application context as a CDI bean

The Seam Spring extension can install a web application context (the application context created by a ContextLoaderListener) by defining a producer field, as follows:

```
package org.jboss.seam.spring.test.bootstrap;

import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Produces;
```

```
import org.jboss.seam.spring.context.SpringContext;
import org.jboss.seam.spring.context.Web;
import org.springframework.context.ApplicationContext;

public class WebContextProducer {

   @Produces
   @SpringContext
   @Web
   ApplicationContext context;

}
```

Note

The example above will work only in a web application with a Spring application context boostrapped by a ContextLoaderListener.

The `@org.jboss.seam.spring.context.Web` annotation must be placed only on the producer field for the `ApplicationContext`, and it will register a producer that looks up the parent web `ApplicationContext`.

3.1.1.2.2. Installing a custom-configured Spring application context

The Seam Spring extension can create a Spring application ad-hoc and install it as a Spring context as follows:

```
package org.jboss.seam.spring.test.bootstrap;

import javax.enterprise.inject.Produces;

import org.jboss.seam.spring.context.Configuration;
import org.jboss.seam.spring.bocontextpringContext;
import org.springframework.context.ApplicationContext;

public class ConfigurationContextProducer {

  @Produces
  @SpringContext
            @Configuration(locations = "classpath*:org/jboss/seam/spring/test/bootstrap/
applicationContext.xml")
  ApplicationContext context;
```

```
}
```

The `@org.jboss.seam.spring.context.Configuration` annotation must be placed only on the producer field of the `ApplicationContext`, and it will register a producer that creates an `ApplicationContext` from the files in the locations attribute of the annotation.

The attributes supported by @org.jboss.seam.spring.bootstrap.Configuration are listed in the following table:

Table 3.2. Attributes of the @org.jboss.seam.spring.context.Configuration

Attribute	Type	Significance
`locations`	`String`	Comma-separated list of file locations. Observes the conventions regarding the 'classpath:', 'classpath*:' and 'file:' prefixes of Spring

3.1.1.3. Implicit Spring context bootstrapping

The producer fields provide a convenient and accesible way of registering a Spring ApplicationContext, especially for looking up contexts created externally (although direct bootstrap is supported as well). A number of Spring ApplicationContexts can also be created by the extension itself.

This can be done by creating a file named /META-INF/org.jboss.seam.spring.contexts which contains a number of key-value pairs, with the keys representing context names and values representing context locations, as follows:

```
default=classpath*:org/jboss/seam/spring/test/bootstrap/applicationContext.xml
```

Note

The extension supports the registration of multiple application contexts.

An important feature of Spring context bootstrapping is that Spring beans will be automatically vetoed as CDI beans.

The main difference between implicit bootstrapping and producer-field based bootstrapping is that implicit bootstrapping creates the Spring context during CDI deployment and explicit bootstrapping creates a Spring context after deployment. As such, implicit deployment can do various tasks such as auto-vetoing Spring beans and preventing them to be deployed as CDI beans.

3.1.2. Exposing Spring beans as CDI beans

Spring beans can be added as CDI beans explicitly, using a producer field. In order to do so, a Spring ApplicationContext must be registered if they are created by one of the CDI-accessible Spring contexts, as shown in the previous section. This can be done by producer fields and the @SpringBean annotation, as in the following example:

```
public class SimpleBeanProducer {

   @Produces @SpringBean(fromContext = "context2") SimpleBean simpleBean;

   @Produces @SpringBean ComplicatedBean complicatedBean;
}
```

The result is that two CDI beans are available for injection and lookup: one based on the SimpleBean Spring bean defined in the Spring context registered as 'context2' and the other, based on the ComplicatedBean defined in the Spring context registered as 'default'.

3.2. Importing CDI beans into Spring applications

The Seam Spring module also supports the registration of CDI beans as Spring beans as well. Once CDI beans are imported into a Spring ApplicationContext, they can be injected as regular Spring beans, either via XML or by annotations.

This can be done by using the dedicated CDI namespace, which can be defined as in the following example:

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:cdi="http://www.jboss.org/schema/seam/spring"
                  xsi:schemaLocation="http://www.springframework.org/schema/beans    http://
www.springframework.org/schema/beans/spring-beans.xsd
       http://www.jboss.org/schema/seam/spring http://www.jboss.org/schema/seam/spring/seam-
spring.xsd">

   <!-- bean definitions -->

</beans>
```

3.2.1. Registering a BeanManager

Spring applications can get access to a BeanManager through the following bean definition.

```
<cdi:bean-manager/>
```

The bean has the id 'beanManager' by default.

3.2.2. Importing a CDI bean as a Spring bean

A CDI bean can be imported as a Spring bean by using a namespace element as follows:

```
<cdi:bean-reference id="cdiBean" type="org.jboss.seam.spring.test.injection.CdiBean"/>
```

A CDI bean with qualfiers can be imported as follows:

```
<cdi:bean-
reference id="secondCdiBean" type="org.jboss.seam.spring.test.injection.SecondCdiBean">
  <cdi:qualifier type="org.jboss.seam.spring.test.injection.CdiQualifier"/>
</cdi:bean-reference>
```

If the qualifiers have attributes, the bean can be imported as follows:

```
<cdi:bean-
reference id="thirdCdiBean" type="org.jboss.seam.spring.test.injection.ThirdCdiBean">
  <cdi:qualifier type="org.jboss.seam.spring.test.injection.CdiQualifierWithAttributes">
    <cdi:attribute name="name" value="myBean"/>
  </cdi:qualifier>
 </cdi:bean-reference>
```

The conversion from String to the actual type of the attribute is handled by Spring's ConversionService.

CDI beans are imported as prototype-scoped Spring beans, which means that a new reference is acquired every time the bean is injected into a Spring bean. This is done in order to preserve the original scope of the CDI bean.