Basics

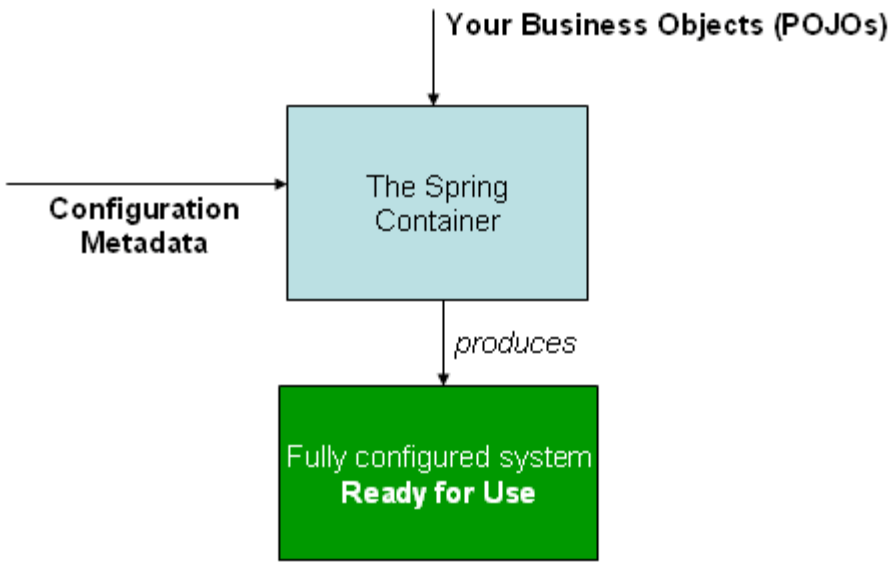
Summary

Description

The Bean in the Spring Framework composes the application and refers to the instance that is managed by IoC container. In short, the Bean is instantiated, assembled and managed by the IoC container. The Bean and dependency between the Beans is decided by the configuration metadata that the container uses.

The container

The org.springframework.beans.factory.BeanFactory is the key interface of the Spring IoC Container. It creates the instance and connects the dependency between instances.



Configuration metadata

As seen above, the Spring IoC container requires configuration metadata. This configuration metadata provides necessary information for the Spring IoC container to "instantiate, configure and assemble the objects." This configuration metadata is typically supplied in a simple and intuitive XML format. You can find details of another form of metadata that the Spring container can consume in the section entitled "Annotation-based configuration"

Find below an example of the basic structure of XML-based configuration metadata.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
        xsi:schemaLocation="http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
  </bean>

  <bean id="..." class="...">
    <!-- collaborators and configuration for this bean go here -->
  </bean>

  <!-- more bean definitions go here -->

</beans>
```

<beans> element is the element that displays the configuration metadata of the Spring IoC container. Moreover, each <bean> element indicates the definition of objects that the Spring IoC container creates and manages.

The XML-based configuration metadata can be composed of many files. In this case, other files can be imported from one configuration file to read overall configuration metadata. Use the <import> element to import.

```
<beans>

    <import resource="services.xml"/>
    <import resource="resources/messageSource.xml"/>
    <import resource="/resources/themeSource.xml"/>

    <bean id="bean1" class="..."/>
    <bean id="bean2" class="..."/>

</beans>
```

The resource attribute of <import> tag indicates the location of XML configuration file.

Instantiating a container

Next is an example of instantiating the Container.

```
ApplicationContext context = new ClassPathXmlApplicationContext(
        new String[] {"services.xml", "daos.xml"});

// an Application is also a BeanFactory (via inheritance)
BeanFactory factory = context;
```

ClassPathXmlApplicationContext of the example above is a type of ApplicationContext and the ApplicationContext interface inherits the BeanFactory interface.

How to Use the Container

When the container is objectification, the bean can be imported using the get Bean (String) method.

The beans

Spring IoC container manages many beans. The container uses the configuration metadata to create beans. The definition of beans used in the container conveys below data.

- A package-qualified class name: typically this is the actual implementation class of the bean being defined.
- Bean behavioral configuration elements, which state how the bean should behave in the container (scope, lifecycle callbacks, and so forth).
- References to other beans which are needed for the bean to do its work; these references are also called *collaborators* or *dependencies*.
- Other configuration settings to set in the newly created object. An example would be the number of connections to use in a bean that manages a connection pool, or the size limit of the pool.

The concepts listed above directly translate to a set of properties that each bean definition consists of. Some of these properties are listed below, along with a link to further documentation about each of them.

Feature	Explained in...
class	Instantiation beans
name	Naming beans
scope	Bean scope
constructor arguments	Injecting dependencies
properties	Injecting dependencies
autowiring mode	Autowiring collaborators
dependency checking mode	Checking for dependencies)
lazy-initialization mode	Lazily-instantiated beans
initialization method	Initialization callbacks
destruction method	Destruction callbacks

Naming beans

All beans should have more than one ids. Only one id is permitted within the container. A bean will almost always have only one id, but if a bean has more than one id, the extra ones can essentially be considered aliases.

The convention (at least amongst the Spring development team) is to use the standard Java convention for instance field names when naming beans. That is, bean names start with a lowercase letter, and are camel-cased from then on. Examples of such names would be (without quotes) 'accountManager', 'accountService', 'userDao', 'loginController', and so forth.

Aliasing beans

Additional names can be given to the pre-defined beans using <alias> element.

<alias name="fromName" alias="toName"/>

The name attribute is the name of the bean and the alias attribute is the new name to be given.

Instantiating beans

All definition of beans requires Java Class for Instantiation.

The class property specifies the class of the bean to be constructed in the common case where the container itself directly creates the bean by calling its constructor reflectively (somewhat equivalent to Java code using the *'new'* operator). In the less common case where the container invokes a static, *factory* method on a class to create the bean, the class property specifies the actual class containing the static factory method that is to be invoked to create the object

The instantiation using the creator is most common and it is used as following.

<bean id="exampleBean" class="example.ExampleBean"/>

<bean name="anotherExample" class="examples.ExampleBeanTwo"/>

Reference

- Spring Framework - Reference Document / 3.2. Basics - containers and beans