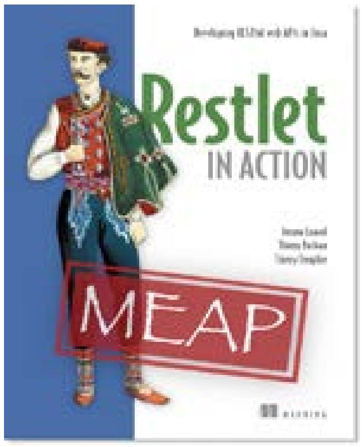
MANNING PUBLICATIONS



Declarative Configuration in XML

In this article, we will explain how to obtain equivalent component configurations using a declarative XML approach instead of plain Java code. We will see examples using the simple XML format supported by the `Component` class, then the more complete and powerful approach based on the Spring Framework.

XML configuration with Component class

The first way to use XML is built into the `Component` class itself, allowing you to use a simple XML document to configure your connectors, virtual hosts, and other properties that we use in the `MailServerComponent` class. In listing 1, we illustrate how we can fully replace the `MailServerComponent` class by a single XML document.

Listing 1 Declarative XML configuration with the Component class

```
<?xml version="1.0"?>
<component xmlns="http://www.restlet.org/schemas/2.0/Component"          #A
     name="RESTful Mail Server component"
     description="Example for 'Restlet in Action' book"
     owner="Noelios Technologies"
     author="The Restlet Team">

    <client protocol="CLAP" />                                           #B

    <server protocol="HTTP">
        <parameter name="tracing" value="true" />
    </server>

    <!--
    <defaultHost                                                 #C
           hostDomain="www.rmep.com|www.rmep.net|www.rmep.org"
           serverAddress="1.2.3.10|1.2.3.20" serverPort="80">
        <attachDefault
            targetClass="org.restlet.example.book.restlet
                  [CA].ch04.sec3.server.MailServerApplication" />
    </defaultHost>
    -->

    <defaultHost>
        <attachDefault
            targetClass="org.restlet.example.book.restlet
```

```
                [CA].ch04.sec3.server.MailServerApplication" />
    </defaultHost>

    <logService loggerName="MailServer.AccessLog"                       #D
            logPropertiesRef="clap://system/org/restlet/example/
                 [CA]book/restlet/ch04/sec3/server/log.properties" />

</component>
```

#A Sets basic properties
#B Declares connectors
#C Declares virtual hosts
#D Configuresa scwimproves application maintainability.

When using dependency injection, a lightweight container like Spring is required that has the responsibilities to manage these POJOs and link them. Metadata are required to specify bean dependencies using for instance XML files or annotations.

SPRING FRAMEWORK

Spring is a layered application framework and lightweight container, which foundations are exposed in the book *Expert One-on-One J2EE Design and Development* by Rod Johnson. The Spring project began in 2003. The lightweight container and the aspect-oriented programming (AOP) system are the main building blocks of Spring. Beside them, Spring comes with a common abstraction layer for transaction management, integration with various persistence solutions (plain JDBC, Hibernate, JPA) as well as with Java enterprise technologies (JMS, JMX). Spring even provides its own Model View Controller (MVC) Web framework, Spring MVC. The Spring framework is not an “all-or-nothing” solution: one can choose the modules according to their needs, the lightweight container being the glue for the application and the Spring classes.

To make easier the use of Restlet with the Spring framework, in addition to the getter and setter methods available for most of the Restlet API classes, a special extension is provided in the “org.restlet.ext.spring” module. It contains specialized classes to facilitate the configuration of the parent class from the Spring Framework, for example you can use `SpringComponent` instead of its parent `Component` class.

Let’s now see in listing 2 how we can configure a component equivalent to our `MailServerComponent` using Spring XML.

Listing 2 Declarative XML configuration with the Spring Framework

```
<?xml version="1.0"?>
<beans xmlns="http://www.springframework.org/schema/beans"              #A
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
         http://www.springframework.org/schema/util
           http://www.springframework.org/schema/util/spring-util-3.0.xsd">

    <bean id="component"                                                #B
          class="org.restlet.ext.spring.SpringComponent">
        <property name="name" value="RESTful Mail Server component" />
        <property name="description"
                  value="Example for 'Restlet in Action' book" />
        <property name="owner" value="Noelios Technologies" />
        <property name="author" value="The Restlet Team" />
        <property name="client" value="clap" />
        <property name="server" ref="server" />
        <property name="defaultHost" ref="defaultHost" />
    </bean>

    <bean id="component.context"                                        #C
         class="org.springframework.beans.factory
                  [CA].config.PropertyPathFactoryBean" />

    <bean id="server" class="org.restlet.ext.spring.SpringServer">       #D
        <constructor-arg value="http" />
        <constructor-arg value="8111" />
        <property name="parameters">
            <props>
                <prop key="tracing">true</prop>
            </props>
        </property>
    </bean>

    <bean id="defaultHost" class="org.restlet.ext.spring.SpringHost">    #E
        <constructor-arg ref="component" />

        <!--
        <property name="hostDomain"
                  value="www.rmep.com|www.rmep.net|www.rmep.org" />
        <property
                  name="serverAddress" value="1.2.3.10|1.2.3.20" />
        <property
                  name="serverPort" value="80" />
        -->

        <property name="defaultAttachment" ref="mailServerApplication" />
    </bean>
</beans>
```

#A Root element with namespaces declarations
#B Component properties set by value or reference
#C Extraction of component's context for reuse
#D HTTP server with tracing enabled
#E Virtual host with attached mail server application

As you can see, each instance of bean managed by Spring is defined using a bean XML element. It allows specifying both corresponding class (class attribute) of the instance and its properties (properties XML element). The property XML element can either contain value or a reference to another bean managed by Spring.With the above configuration, we have declared the equivalent of our existing `MailServerComponent`. Spring allows using each Restlet's element to assembly the component. We now want to illustrate how with the Spring Framework we can go beyond and configure other parts of the Restlet API, such as an application in its contained resources. Listing 3 contains the continuation of the previous listing.

Listing 3 Declarative XML configuration with the Spring Framework (continued)

```
    <bean id="componentChildContext" class="org.restlet.Context">        #A
        <lookup-method name="createChildContext"
                       bean="component.context" />
    </bean>

    <bean id="mailServerApplication" class="org.restlet.Application">    #B
        <constructor-arg ref="componentChildContext" />

        <property name="name" value="RESTful Mail Server application"/>
        <property name="description"
              value="Example application for 'Restlet in Action' book" />
        <property name="owner" value="Noelios Technologies" />
        <property name="author" value="The Restlet Team" />
        <property name="inboundRoot">
            <bean class="org.restlet.ext.spring.SpringRouter">
                <constructor-arg ref="mailServerApplication" />
                <property name="attachments">                            #C
                    <map>
                        <entry key="/"
                           value="org.restlet.example.book.restlet.
                             [CA]ch04.sec3.server.RootServerResource" />
                        <entry key="/accounts/"
                           value="org.restlet.example.book.restlet.
                             [CA]ch04.sec3.server.AccountsServerResource" />
                        <entry key="/accounts/{accountId}"
                           value="org.restlet.example.book.restlet.
                             [CA]ch04.sec3.server.AccountServerResource" />
                    </map>
                </property>
            </bean>
        </property>
    </bean>
</beans>
```
#A Create child context
#B Application bean referenced by default host
#C Convenient way to declare routes with SpringRouter

This part of the configuration is a bit verbose since there are a lot of hints to specify on the application. First, the child context and general application information are set. You then specify how to access your resources thanks to a `SpringRouter` instance and its `attachments` property. This router is injected within the application using its `inboutRoot` property. This part corresponds to what you specify in the `createInboundRoot` method without Spring.

Note that we had to manually create a child context as the component's context can't be directly passed to isolate all contained applications, for security reasons. With the Java API, this child context is automatically created by the attachment methods, but with Spring it has to be explicitly done at instantiation time.

As a result, we were able to configure both our component and our application in XML with Spring, leaving us to just code the server resources. This can be useful in situations where you want to provide different sets of resources for an application based on some deployment aspects, or different sets of applications for a component.

Of course, this is just an overview of the features offered by the Spring extension for Restlet. You can check the Javadocs and the online user guide for additional aspects. Finally, let's make sure that our Spring XML configuration works, by running the small bootstrap code in listing 4 below.

Listing 4 Running the declarative XML configuration with the Spring Framework

```
public class MailServerSpring {

    public static void main(String[] args) throws Exception {
        // Load the Spring container
        ClassPathResource resource = new ClassPathResource(
                "org/restlet/example/book/restlet/ch03
                    [CA]/sec3/server/component-spring.xml");
        BeanFactory factory = new XmlBeanFactory(resource);

        // Start the Restlet component
        Component component = (Component) factory.getBean("component");
        component.start();
    }

}
```
The `BeanFactory` entity represents the Spring container and can be loaded using several strategies. In the previous listing we loaded from the classpath basing on the `ClassPathResource` class. The later looks for the file in the classpath and the specified path is relative to the classpath root. Since you use an XML file for metadata, the `XmlBeanFactory` class is used. Now that the container is loaded, you can access configured beans by identifier using the getBean method. In the sample, we get the Restlet component instance and start it with its `start` method.

In this article, we saw alternative ways to leverage the Restlet API and especially to configure Restlet components, using XML instead of Java code. We leverage the built-in XML configuration capabilities of the Component class and more advanced XML configuration based on the Spring Framework. Additional approaches have been successfully explored by Restlet community members, such as the usage of other inversion of control frameworks like Google Guice or dynamic languages such as Groovy and Scala.

As underlined previously in the article, Spring XML configurations can be verbose due to XML itself. To address this issue, Spring provides the ability to use dedicated XML schemas for particular use cases. The most well-known ones are the ones for AOP and transactions. With them, you don't need any more to know the underlying beans implemented. The schema hides this complexity. The Restlet extension for Spring doesn't provide such schema yet but an attempt has done. See the Restlet issue #638 for more details.

Listing 5 Configuration using Spring namespace for Restlet

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

    <restlet:application id="application">
        <restlet:attachments>
            <restlet:attachment route="/companies/{id}"
                                resource="companyResource"/>
        </restlet:attachments>
    </restlet:application>
</beans>
```

After having configured the Restlet namespace, you can use the dedicated XML grammar it provides to your Restlet application easier.

Summary

We described two alternative deployment mechanisms useful for administrators that can’t recompile Java code to set things specific to deployment environments such as a port number or a domain name. In those cases, declarative XML configuration files can be used, leveraging the Spring Framework if necessary.

Last updated: January 25, 2012