Appendix

Version 5.1.2.RELEASE

Chapter 1. XML Schemas

This part of the appendix lists XML schemas related to integration technologies.

1.1. The jee Schema

The jee elements deal with issues related to Java EE (Java Enterprise Edition) configuration, such as looking up a JNDI object and defining EJB references.

To use the elements in the jee schema, you need to have the following preamble at the top of your Spring XML configuration file. The text in the following snippet references the correct schema so that the elements in the jee namespace are available to you:

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

    <!-- bean definitions here -->
</beans>
```

1.1.1. <jee:jndi-lookup/> (simple)

The following example shows how to use JNDI to look up a data source without the jee schema:

```
<bean id="<strong>dataSource</strong>"
class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="jdbc/MyDataSource"/>
</bean>
<bean id="userDao" class="com.foo.JdbcUserDao">
    <!-- Spring will do the cast automatically (as usual) -->
    <property name="dataSource" ref="<strong>dataSource</strong>"/>
</bean>
```

The following example shows how to use JNDI to look up a data source with the jee schema:

```
<jee:jndi-lookup id="<strong>dataSource</strong>" jndi-name="jdbc/MyDataSource"/>

<bean id="userDao" class="com.foo.JdbcUserDao">
    <!-- Spring will do the cast automatically (as usual) -->
    <property name="dataSource" ref="<strong>dataSource</strong>"/>
</bean>
```

1.1.2. <jee:jndi-lookup/> (with Single JNDI Environment Setting)

The following example shows how to use JNDI to look up an environment variable without jee:

```
<bean id="simple" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="jdbc/MyDataSource"/>
    <property name="jndiEnvironment">
        <props>
            <prop key="ping">pong</prop>
        </props>
    </property>
</bean>
```

The following example shows how to use JNDI to look up an environment variable with jee:

```
<jee:jndi-lookup id="simple" jndi-name="jdbc/MyDataSource">
    <jee:environment>ping=pong</jee:environment>
</jee:jndi-lookup>
```

1.1.3. <jee:jndi-lookup/> (with Multiple JNDI Environment Settings)

The following example shows how to use JNDI to look up multiple environment variables without jee:

```
<bean id="simple" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="jdbc/MyDataSource"/>
    <property name="jndiEnvironment">
        <props>
            <prop key="sing">song</prop>
            <prop key="ping">pong</prop>
        </props>
    </property>
</bean>
```

The following example shows how to use JNDI to look up multiple environment variables with `jee`:

```
<jee:jndi-lookup id="simple" jndi-name="jdbc/MyDataSource">
    <!-- newline-separated, key-value pairs for the environment (standard
Properties format) -->
    <jee:environment>
        sing=song
        ping=pong
    </jee:environment>
</jee:jndi-lookup>
```

1.1.4. `<jee:jndi-lookup/>` (Complex)

The following example shows how to use JNDI to look up a data source and a number of different properties without `jee`:

```
<bean id="simple" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="jdbc/MyDataSource"/>
    <property name="cache" value="true"/>
    <property name="resourceRef" value="true"/>
    <property name="lookupOnStartup" value="false"/>
    <property name="expectedType" value="com.myapp.DefaultThing"/>
    <property name="proxyInterface" value="com.myapp.Thing"/>
</bean>
```

The following example shows how to use JNDI to look up a data source and a number of different properties with `jee`:

```
<jee:jndi-lookup id="simple"
        jndi-name="jdbc/MyDataSource"
        cache="true"
        resource-ref="true"
        lookup-on-startup="false"
        expected-type="com.myapp.DefaultThing"
        proxy-interface="com.myapp.Thing"/>
```

1.1.5. `<jee:local-slsb/>` (Simple)

The `<jee:local-slsb/>` element configures a reference to a local EJB Stateless SessionBean.

The following example shows how to configures a reference to a local EJB Stateless SessionBean without `jee`:

```
<bean id="simple"
        class=
"org.springframework.ejb.access.LocalStatelessSessionProxyFactoryBean">
    <property name="jndiName" value="ejb/RentalServiceBean"/>
    <property name="businessInterface" value="com.foo.service.RentalService"/>
</bean>
```

The following example shows how to configures a reference to a local EJB Stateless SessionBean with `jee`:

```
<jee:local-slsb id="simpleSlsb" jndi-name="ejb/RentalServiceBean"
        business-interface="com.foo.service.RentalService"/>
```

1.1.6. `<jee:local-slsb/>` (Complex)

The `<jee:local-slsb/>` element configures a reference to a local EJB Stateless SessionBean.

The following example shows how to configures a reference to a local EJB Stateless SessionBean and a number of properties without `jee`:

```
<bean id="complexLocalEjb"
        class=
"org.springframework.ejb.access.LocalStatelessSessionProxyFactoryBean">
    <property name="jndiName" value="ejb/RentalServiceBean"/>
    <property name="businessInterface" value="com.example.service.RentalService"/>
    <property name="cacheHome" value="true"/>
    <property name="lookupHomeOnStartup" value="true"/>
    <property name="resourceRef" value="true"/>
</bean>
```

The following example shows how to configures a reference to a local EJB Stateless SessionBean and a number of properties with `jee`:

```
<jee:local-slsb id="complexLocalEjb"
        jndi-name="ejb/RentalServiceBean"
        business-interface="com.foo.service.RentalService"
        cache-home="true"
        lookup-home-on-startup="true"
        resource-ref="true">
```

1.1.7. <jee:remote-slsb/>

The `<jee:remote-slsb/>` element configures a reference to a `remote` EJB Stateless SessionBean.

The following example shows how to configures a reference to a remote EJB Stateless SessionBean without `jee`:

```
<bean id="complexRemoteEjb"
        class=
"org.springframework.ejb.access.SimpleRemoteStatelessSessionProxyFactoryBean">
    <property name="jndiName" value="ejb/MyRemoteBean"/>
    <property name="businessInterface" value="com.foo.service.RentalService"/>
    <property name="cacheHome" value="true"/>
    <property name="lookupHomeOnStartup" value="true"/>
    <property name="resourceRef" value="true"/>
    <property name="homeInterface" value="com.foo.service.RentalService"/>
    <property name="refreshHomeOnConnectFailure" value="true"/>
</bean>
```

The following example shows how to configures a reference to a remote EJB Stateless SessionBean with `jee`:

```
<jee:remote-slsb id="complexRemoteEjb"
        jndi-name="ejb/MyRemoteBean"
        business-interface="com.foo.service.RentalService"
        cache-home="true"
        lookup-home-on-startup="true"
        resource-ref="true"
        home-interface="com.foo.service.RentalService"
        refresh-home-on-connect-failure="true">
```

1.2. The `jms` Schema

The `jms` elements deal with configuring JMS-related beans, such as Spring's Message Listener Containers. These elements are detailed in the section of the JMS chapter entitled JMS Namespace Support. See that chapter for full details on this support and the `jms` elements themselves.

In the interest of completeness, to use the elements in the `jms` schema, you need to have the following preamble at the top of your Spring XML configuration file. The text in the following snippet references the correct schema so that the elements in the `jms` namespace are available to you:

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

    <!-- bean definitions here -->
</beans>
```

1.3. Using <context:mbean-export/>

This element is detailed in Configuring Annotation-based MBean Export.

1.4. The cache Schema

You can use the cache elements to enable support for Spring's @CacheEvict, @CachePut, and @Caching annotations. It it also supports declarative XML-based caching. See Enabling Caching Annotations and Declarative XML-based Caching for details.

To use the elements in the cache schema, you need to have the following preamble at the top of your Spring XML configuration file. The text in the following snippet references the correct schema so that the elements in the cache namespace are available to you:

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

    <!-- bean definitions here -->
</beans>
```