Spring

Sesión 4: Acceso remoto a beans

Indice

- Spring vs Contenedor EJB para proveer servicios
- Acceso remoto en Spring
- Transaccionalidad declarativa en Spring

Spring ofrece servicios a los beans

- Hasta ahora el único que hemos usado ha sido el de instanciarlos y mantener las dependencias
- Los EJBs son apreciados sobre todo porque liberan al programador de la carga de implementar ciertos servicios:
  - Acceso remoto a clientes no web
  - Seguridad declarativa
  - Transaccionalidad declarativa
- Spring ofrece el 80% de la funcionalidad de los EJBs con un 20% de su complejidad

Acceso remoto en Spring

- Hacer accesibles nuestros métodos y clases desde otras máquinas (no solo a través de la capa web)
- ¿Por qué acceso remoto?
  - Clientes ricos (Swing, etc)
  - Servicios web
  - Aplicaciones distribuidas
- Spring tiene ciertas limitaciones, relativas sobre todo a transacciones que impliquen objetos remotos. Para eso necesitaremos EJBs

Opciones en Spring para acceso remoto

- RMI
  - Clientes Java, eficiente, problemas con firewalls
- Hessian y Burlap
  - Clientes en varios lenguajes, eficiencia media, *firewall friendly*
- HTTP invoker
  - Clientes Spring, eficiencia media, *firewall friendly*
- Servicios web
  - Estandarización, portabilidad máxima en cuanto a clientes, poca eficiencia, *firewall friendly*

Bean en el servidor

- Queremos acceder remotamente a él ¿Qué hacer?

Java

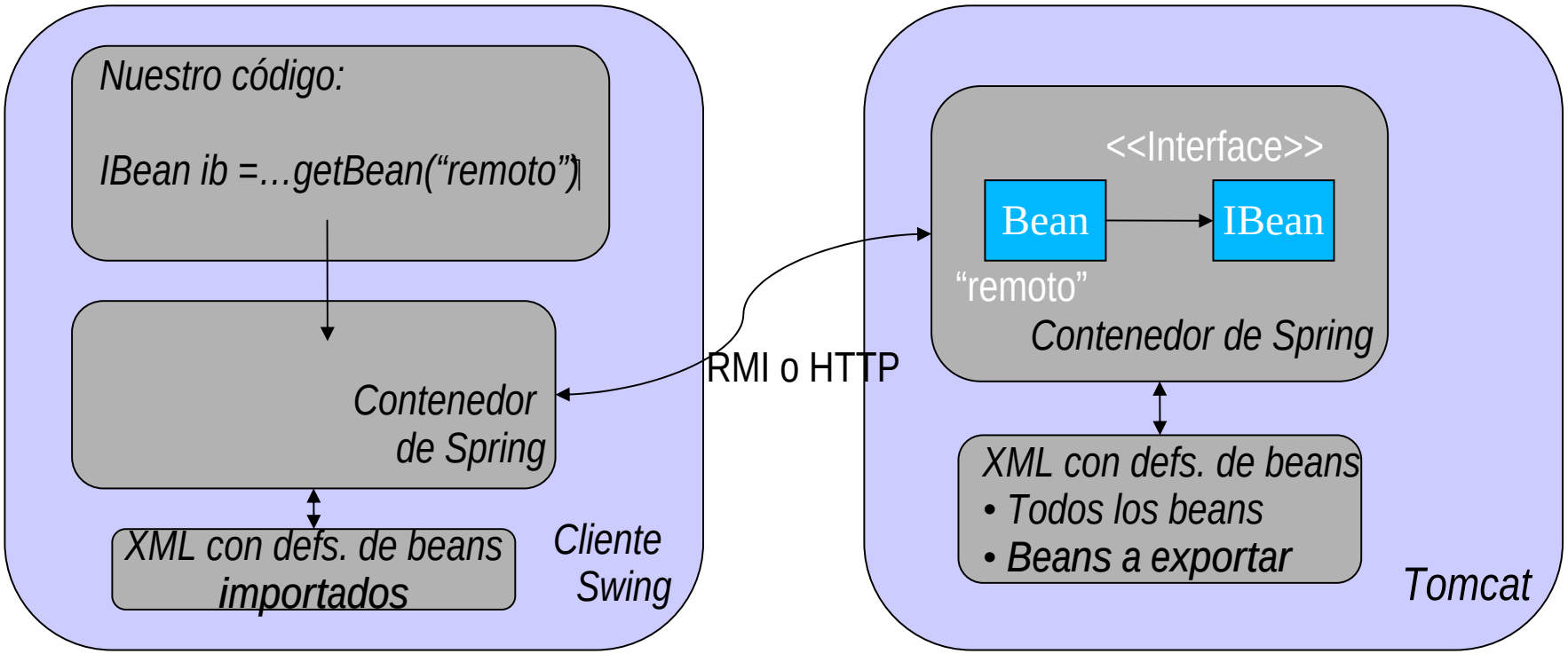
```
package servicios;

public interface ServicioSaludo {
  public String getSaludo();
}
```

```
package servicios;

public class ServicioSaludoImpl implements ServicioSaludo {
  String[] saludos = {"hola, ¿qué tal?", "me alegra verte", "yeeeeeey"};

  public String getSaludo() {
      int pos = (int)(Math.random() * saludos.length);
      return saludos[pos];
  }
}
```

XML con beans

```
<bean id="miSaludador" class="servicios.ServicioSaludoImpl">
</bean>
```

Esquema



Acceso remoto con RMI

- RMI es el protocolo estándar de Java para la ejecución remota de objetos
- Tiene fama de ser complicado, por ello en Spring no se usa directamente, sino con una capa de abstracción por encima
- Los puertos son distintos a HTTP (problemas con *firewalls*)

RMI en el servidor

- Definir un bean de la clase RmiServiceExporter

```
<bean class="org.springframework.remoting.rmi.RmiServiceExporter">
   <!-- En este caso el nombre del servicio y del bean son iguales, pero no tiene por que -->
   <property name="serviceName" value="miSaludador"/>
   <property name="service" ref="miSaludador"/>
   <property name="serviceInterface" value="servicios.ServicioSaludo"/>
   <!-- El puerto por defecto es el 1099 -->
   <property name="registryPort" value="1199"/>
</bean>
```

- La URL del servicio será
  - rmi://localhost:1199/miSaludador

RMI en el cliente

- Llamar al bean remoto con Java:

```
ClassPathXmlApplicationContext contexto = new
                    ClassPathXmlApplicationContext("clienteRMI.xml");
ServicioSaludo ss =  (ServicioSaludo) contexto.getBean("saludadorRMI");
System.out.println(ss.getSaludo());
```

- Fichero “clienteRMI.xml”

```
<bean id="saludador" class="org.springframework.remoting.rmi.RmiProxyFactoryBean">
    <property name="serviceUrl" value="rmi://localhost:1199/miSaludador"/>
    <property name="serviceInterface" value=“servicios.ServicioSaludo"/>
</bean>
```

Hessian y Burlap

- Protocolos desarrollados por la empresa Caucho (www.caucho.com) para
  - Funcionar con HTTP
  - Ser razonablemente eficientes
  - Portabilidad en el cliente: hay librerías para C#, Python, C++, PHP,…
- Hessian es binario y Burlap XML, por lo demás la idea es similar al menos desde el punto de vista de la configuración necesaria en Spring.

Hessian en Spring. Parte servidor

- La comunicación con el exterior se hace a través de un servlet. Usaremos el DispatcherServlet que ya usábamos en MVC

```
<servlet>
        <servlet-name>remoting</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>remoting</servlet-name>
        <url-pattern>/remoting/*</url-pattern>
</servlet-mapping>
```

Fragmento del web.xml

- Ahora tenemos que exportar el bean

```
<bean name="/saludador"
    class="org.springframework.remoting.caucho.HessianServiceExporter">
  <property name="service" ref="miSaludador"/>
  <property name="serviceInterface" value="servicios.ServicioSaludo"/>
</bean>
```

Fichero remoting-servlet.xml, en WEB-INF

- El servicio estará en http://localhost:8080/*contexto-web*/remoting/saludador

Hessian en el cliente

- Llamar al bean remoto con Java:

```
ClassPathXmlApplicationContext contexto = new
    ClassPathXmlApplicationContext("clienteHessian.xml");
ServicioSaludo ss =  (ServicioSaludo) contexto.getBean("miSaludador");
System.out.println(ss.getSaludo());
```

- Fichero “clienteHessian.xml”

```
<bean id="miSaludador"
      class="org.springframework.remoting.caucho.HessianProxyFactoryBean">
      <property name="serviceUrl"
            value="http://localhost:8080/contexto-web/remoting/saludador"/>
      <property name="serviceInterface" value="servicios.ServicioSaludo"/>
</bean>
```

Hessian permite autentificación declarativa

```
<!-- este bean de Spring es el que asocia la URL /saludador con el bean llamado ídem-->
<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
  <property name="interceptors">
    <list>
      <ref bean="authorizationInterceptor"/>
    </list>
  </property>
</bean>

<bean id="authorizationInterceptor"
  class="org.springframework.web.servlet.handler.UserRoleAuthorizationInterceptor">
  <property name="authorizedRoles">
   <list>
    <value>admin</value>
    <value>subadmin</value>
   </list>
  </property>
</bean>
```

HTTP invoker

- Parecido a Hessian pero propio de Spring
  - Ventaja: el mecanismo de serialización es más potente que el de Hessian, que puede fallar en casos complejos
  - Inconveniente: el cliente debe ser Java y además Spring
- Servidor
  - Se usa el mismo DispatcherServlet que con Hessian (mismo web.xml)
  - Solo falta exportar el bean

```
<bean name="/saludadorHTTP"
    class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service" ref="miSaludador"/>
  <property name="serviceInterface" value="servicios.ServicioSaludo"/>
</bean>
```

Fichero remoting-servlet.xml, en WEB-INF

HTTP invoker en el cliente

- Llamar al bean remoto desde Java

```
ClassPathXmlApplicationContext contexto = new
                    ClassPathXmlApplicationContext("clienteHTTP.xml");
ServicioSaludo ss =  (ServicioSaludo) contexto.getBean("httpProxy");
System.out.println(ss.getSaludo());
```

- Fichero “clienteHTTP.xml”

```
<bean id="httpProxy"
        class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
        <property name="serviceUrl"
                value="http://localhost:8080/CONTEXTO-WEB/remoting/saludadorHTTP"/>
        <property name="serviceInterface" value="servicios.ServicioSaludo"/>
</bean>
```

¿Preguntas...?