

Remoting with Spring
From Hessian to RMI

Jürgen Höller

<http://www.springframework.com>

juergen@interface21.com

Agenda

- Remoting
- Hessian and Burlap
- HTTP invoker
- RMI

Introduction

- "Rich Clients"
 - remote application that accesses backend server
 - "rich" user interface: Java Web Start, Flash
- Integration needs
 - remote processes that access backend server
 - to be added to existing web application
- Spring's remoting support
 - exposure of beans as HTTP-based remote services
 - e.g. Hessian, Burlap, HTTP invoker
 - integration with Web Services via JAX-RPC
 - e.g. Apache Axis

Remoting (1)

- Different needs
 - stateless remoting vs stateful remoting?
 - multiple platforms or Java-only?
 - remote transaction propagation?
- Different technologies
 - Remote EJB
 - *Hessian and Burlap*
 - *HTTP invoker*
 - *RMI*
 - *WSDL Web Services (via JAX-RPC)*

Remoting (2)

- Spring provides remoting abstraction
 - generic unchecked RemoteAccessException
 - support classes for access and export of services
 - goal: allow any business interface for remote service
- Pre-built integration classes
 - Remote EJB accessors
 - *Hessian and Burlap support*
 - *HTTP invoker*
 - *RMI support (traditional + RMI invoker)*
 - *JAX-RPC support*

Hessian and Burlap (1)

- Slim protocols for simple needs
 - binary and slim XML, respectively
 - platform-independent, but focus on Java
- HTTP-based web services
 - exposure via provided servlets
 - alternative: programmatic exposure
- Open source projects at Caucho
 - Apache license
 - side projects of Resin application server

Hessian and Burlap (2)

- Spring-style access to remote services
 - make proxy available as bean
 - Hessian/BurlapProxyFactoryBean
 - use a plain business interface!

```
<bean id="hessianProxy"  
class="org.springframework.remoting.caucho.HessianProxyFactoryBean">  
  <property name="serviceUrl">  
    <value>http://localhost:8080/caucho/OrderService-hessian</value>  
  </property>  
  <property name="serviceInterface">  
    <value>  
      org.springframework.samples.jpstore.domain.logic.OrderService  
    </value>  
  </property>  
</bean>
```

Hessian and Burlap (3)

- Spring-style exposure of remote services
 - service exporters defined as beans
 - Hessian/BurlapServiceExporter
 - export an existing Spring-managed bean!

```
<bean name="/OrderService-hessian"  
class="org.springframework.remoting.caucho.HessianServiceExporter">  
  <property name="service">  
    <ref bean="petStore"/>  
  </property>  
  <property name="serviceInterface">  
    <value>  
      org.springframework.samples.jpetsy.domain.logic.OrderService  
    </value>  
  </property>  
</bean>
```

HTTP Invoker (1)

- Java serialization over HTTP
 - special strategy provided by Spring
 - only for Java-to-Java remoting
- Full power of Java serialization
 - just like RMI
- As easy to set up as Hessian or Burlap
 - service identified through HTTP URL
 - no external registry necessary
 - no firewall issues

HTTP Invoker (2)

- Spring-style access to remote services
 - make proxy available as bean
 - HttpInvokerProxyFactoryBean
 - use a plain business interface!

```
<bean id="httpInvokerProxy" class=
"org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
  <property name="serviceUrl">
    <value>http://localhost:8080/caucho/OrderService-
httpinvoker</value>
  </property>
  <property name="serviceInterface">
    <value>
      org.springframework.samples.jpstore.domain.logic.OrderService
    </value>
  </property>
</bean>
```

HTTP Invoker (3)

- Spring-style exposure of remote services
 - service exporters defined as beans
 - `HttpInvokerServiceExporter`
 - export an existing Spring-managed bean!

```
<bean name="/OrderService-httpinvoker" class=
"org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service">
    <ref bean="petStore"/>
  </property>
  <property name="serviceInterface">
    <value>
      org.springframework.samples.jpetsy.domain.logic.OrderService
    </value>
  </property>
</bean>
```

RMI (1)

- Traditional Java remoting
 - full power of Java serialization
 - only for Java-to-Java remoting
- Plain RMI
 - with traditional RMI service interface
- RMI invoker
 - with plain Java business interface
 - special strategy provided by Spring
 - using RMI as backend infrastructure

RMI (2)

- Needs external registry
 - services registered by name
 - service lookup through registry
- Proxies can become stale
 - through restart of the target server
 - refresh on connect failure
 - available in Spring's RMI support
- Performance roughly like HTTP invoker
 - equally fast, but more complex to set up

RMI (3)

- Spring-style access to RMI services
 - RMI URL instead of HTTP URL
 - use a plain business interface even with RMI!

```
<bean id="rmiProxy"  
class="org.springframework.remoting.rmi.RmiProxyFactoryBean">  
  <property name="serviceUrl">  
    <value>rmi://localhost:1099/order</value>  
  </property>  
  <property name="serviceInterface">  
    <value>  
      org.springframework.samples.jpstore.domain.logic.OrderService  
    </value>  
  </property>  
</bean>
```

RMI (4)

■ Spring-style exposure of RMI services

```
<bean id="order-rmi"  
class="org.springframework.remoting.rmi.RmiServiceExporter">  
  <property name="service">  
    <ref bean="petStore"/>  
  </property>  
  <property name="serviceInterface">  
    <value>  
      org.springframework.samples.jpetsy.domain.logic.OrderService  
    </value>  
  </property>  
  <property name="serviceName">  
    <value>order</value>  
  </property>  
  <property name="registryPort">  
    <value>1099</value>  
  </property>  
</bean>
```

JAX-RPC / Axis (1)

- WSDL/SOAP-based Web Services
 - through the standard JAX-RPC API
 - covers client access and server endpoints
- Apache Axis as typical choice
 - open source JAX-RPC implementation
 - not fast, but works
- A lot of configuration necessary
 - due to the abstract nature of SOAP
 - more complex than Hessian and co.

JAX-RPC / Axis (2)

- Spring-style access possible
 - JaxRpcPortProxyFactoryBean
 - many configuration settings
 - JAX-RPC requires RMI service interface
 - Spring can use plain Java interface
- Exposure only via special servlets
 - in case of Axis: AxisServlet
 - endpoint implementations necessary
 - can delegate to Spring-managed beans

Sample Application

- Spring's JPetStore
 - exports OrderService via 5 protocols
 - Hessian, Burlap
 - HTTP invoker
 - RMI (via Spring's RMI invoker mechanism)
 - WSDL/SOAP (via Apache Axis)
 - provides demo client
 - accesses OrderService via all 5 protocols
 - shows response times as rough measure
 - works nicely in plain Tomcat / Jetty / Resin!

Summary (1)

- Spring supports various remoting strategies
 - can expose same service under multiple protocols (at different URLs)
- Consistent usage and configuration style
 - proxies with plain Java service interfaces
 - exporters for existing Spring-managed beans
- Hessian, Burlap, HTTP invoker
 - focus on Java-to-Java remoting
 - simple exposure via web application
 - no need for external registry!

Summary (2)

- SOAP via JAX-RPC / Axis
 - for platform-independent remoting
 - in particular: Java and .NET
 - exposure via web application
 - but complex setup
- *Avoid SOAP for Java-to-Java remoting!*
 - prefer Hessian or HTTP invoker
 - faster and easier to set up
 - expose multiple protocols for different clients

<http://www.springframework.org>
<http://www.springframework.com>

<http://www.caucho.com>
<http://ws.apache.org/axis>