

Anyframe Remoting Plugin



Version 1.0.2

저작권 © 2007-2011 삼성SDS

본 문서의 저작권은 삼성SDS에 있으며 Anyframe 오픈소스 커뮤니티 활동의 목적하에서 자유로운 이용이 가능합니다. 본 문서를 복제, 배포할 경우에는 저작권자를 명시하여 주시기 바라며 본 문서를 변경하실 경우에는 원문과 변경된 내용을 표시하여 주시기 바랍니다. 원문과 변경된 문서에 대한 상업적 용도의 활용은 허용되지 않습니다. 본 문서에 오류가 있다고 판단될 경우 이슈로 등록해 주시면 적절한 조치를 취하도록 하겠습니다.

I. Introduction	1
II. Spring Remoting	2
1. RMI(Remote Method Invocation)	3
1.1. Server Configuration	3
1.1.1. Samples	3
1.2. Client Configuration	4
1.2.1. Samples	4
2. Hessian	5
2.1. Server Configuration	5
2.1.1. Samples	5
2.2. Client Configuration	7
2.2.1. Samples	7
2.3. Hessian과 Burlap의 차이점	7
3. Burlap	8
3.1. Server Configuration	8
3.1.1. Samples	8
3.2. Client Configuration	9
3.2.1. Samples	10
3.3. Hessian과 Burlap의 차이점	10
4. HTTP Invoker	11
4.1. Server Configuration	11
4.1.1. Samples	11
4.2. Client Configuration	13
4.2.1. Samples	13
5. Resources	15

I.Introduction

Remoting Plugin은 Spring Remoting에서 지원하는 원격 기술(RMI, Hessian/Burlap, HTTP Invoker 등) 중 HTTP Invoker의 기본 활용 방법을 가이드하기 위한 샘플 코드와 이를 활용하는데 필요한 참조 라이브러리들로 구성되어 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 remoting-plugin을 설치한다.

```
mvn anyframe:install -Dname=remoting
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

Dependent Plugins

Plugin Name	Version Range
query [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.1.2/reference/htmlsingle/query.html]	2.0.0 > *

II.Spring Remoting

Remoting이란 클라이언트 어플리케이션과 원격 어플리케이션에서 제공하는 서비스간의 의사소통을 말한다.

Spring Remoting에서 지원하는 원격 기술은 다음 표와 같다.

원격 기술	설명
RMI(Remote Method Invocation)	방화벽과 같은 네트워크 제약이 없는 상황에서 자바 기반의 서비스를 공개하거나 접근하려는 경우에 사용한다.
Hessian/Burlap	방화벽과 같이 네트워크 제약이 있는 상황에서 자바 기반의 서비스를 공개하거나 접근하려는 경우에 사용한다.
HTTP Invoker	방화벽과 같이 네트워크 제약이 있는 상황에서 Spring 기반의 서비스를 공개하거나 접근하려는 경우에 사용한다.

• 원격 기술 별 장점 및 단점

원격 기술	장점	단점
RMI	자바 객체 직렬화 매커니즘을 지원하므로 복잡한 형태의 데이터 타입을 네트워크 상에서 전송 가능함	자바 대 자바 Remoting 솔루션이며 HTTP 프로토콜을 사용할 수 없음. RMI 컴파일러와 레지스트리가 별도로 필요함
Hessian/Burlap	방화벽 문제 없음	자체 객체 직렬화 매커니즘을 가지고 있어서 복잡한 형태의 데이터 타입 사용이 어려움. Burlap은 자바 클라이언트만 지원함
HTTP Invoker	HTTP 기반의 Remoting 기술 사용 편의성	자바 대 자바 Remoting 솔루션이며 서버와 클라이언트 모두 Spring 어플리케이션으로 구축해야 함
EJB	Remoting, 어플리케이션 보안, 트랜잭션 처리 등을 위한 J2EE 서비스	무거운(heavyweight) 기술로 J2EE 컨테이너가 반드시 필요함
Web Services	플랫폼과 언어에 독립적	SOAP을 사용하며 웹 서비스 구동을 위한 엔진이 필요함

1.RMI(Remote Method Invocation)

RMI는 JDK 1.1에서 처음으로 자바에 도입되어 자바 프로그램 사이에 통신할 수 있는 방법을 제공하였다. RMI [<http://java.sun.com/j2se/1.5.0/docs/guide/rmi/index.html>] 이전에는 CORBA를 사용하거나 소켓 프로그래밍을 해야만 했었다. 그러나 RMI 서비스를 개발하거나 접근하는 일은 복잡하여 개발하기가 불편하며 다음과 같은 일을 수행해야한다.

- java.rmi.Remote를 상속받은 인터페이스 클래스를 작성한다.
- UnicastRemoteObject를 상속받으며 위의 인터페이스 클래스를 구현하는 구현클래스를 작성한다.
- RMI 컴파일러를 사용하여 클라이언트 Stub 클래스와 서버 Skeleton 클래스를 생성시킨다. (rmic -d classname)
- RMI 레지스트리를 구동시키고 서비스를 레지스트리에 바인딩시킨다.
- 클라이언트 코드를 이용하여 RMI 서비스를 호출하여 사용한다.

Spring의 빈 형태로 서비스를 개발한 후 Spring에서 제공하는 RmiServiceExporter를 사용하면 RMI 객체처럼 서비스 객체의 인터페이스를 쉽게 원격 서비스로 노출할 수 있는 기능을 제공한다. 즉, 위에서 언급한 RMI 서비스 개발 단계에서 수행했던 일들을 RmiServiceExporter에서 수행한다. 개발자는 RMI 서비스와 관련된 개발 작업을 비즈니스 서비스에 반영할 필요가 없으므로 비즈니스 로직에 집중하여 개발할 수 있다. 단, **RMI**는 통신을 위해 특정 **Port**를 사용하므로 방화벽을 통과하기 어렵고, 클라이언트와 서버에서 제공되는 서비스 모두 자바로 작성되어야 한다는 제약사항이 존재한다. 다음은 RMI 기능을 Server와 Client 단에서 어떻게 사용해야 하는지에 대한 사용법이다.

1.1.Server Configuration

서버 구현 방식은 Spring에서 제공하는 org.springframework.remoting.rmi.RmiServiceExporter 클래스를 이용하여 손쉽게 일반 Spring Bean으로 작성된 서비스를 RMI Service로 노출시킬 수 있다.

Property Name	Description	Required	Default Value
serviceName	서비스 이름은 서비스를 RMI Registry에 바인딩 하기 위해 사용된다.	Y	N/A
service	RMI 서비스로 노출시키고 싶은 Spring Bean의 id를 설정한다.	Y	N/A
serviceInterface	RMI 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A
registryPort	RMI 등록(registry)을 위한 port를 오버라이딩하기 위해 사용된다. 작성하지 않은 경우 디폴트로 1099 port가 사용된다.	N	1099

1.1.1.Samples

다음은 RMI 서버 구현 속성 설정에 대한 예제이다. 서비스는 일반 Spring Bean 개발과 동일하며 RmiServiceExporter Bean에서 property 설정 정보를 참조하여 RMI 서비스로 노출시키고 있다.

- **Configuration**

다음은 RMI 서비스를 지원하는 RmiServiceExporter의 속성을 정의한 context-remoting-rmi.xml 의 일부이다.

```
<bean id="org.anyframe.sample.remoting.moviefinder.service.MovieService"
```

```

    class="org.anyframe.sample.remoting.moviefinder.service.impl.MovieServiceImpl">
    <property name="movieDao" ref="movieDao"/>
</bean>

<bean id="movieDao"
    class="org.anyframe.sample.remoting.moviefinder.service.impl.MovieDao" />

<!-- Add RMI ServiceExporter -->
<bean class="org.springframework.remoting.rmi.RmiServiceExporter">

    <property name="serviceName" value="MovieService" />
    <property name="service"
ref="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
    <property name="serviceInterface"
        value="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
    <!-- defaults to 1099 -->
    <property name="registryPort" value="1199" />
</bean>

```

1.2.Client Configuration

클라이언트는 Spring에서 제공하는 `org.springframework.remoting.rmi.RmiProxyFactoryBean` 클래스를 사용하여 RMI Service에 접근할 수 있다.

Property Name	Description	Required	Default Value
serviceUrl	RMI 서비스 접근 URL 정보이다. "rmi://" + 서버 ip + ":" + port 번호 + "/" + 서비스 명 (ex.rmi://localhost:1199/MovieService)	Y	N/A
serviceInterface	RMI 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

1.2.1.Samples

다음은 RMI 클라이언트 속성 설정에 대한 예제이다. 클라이언트는 `RmiProxyFactoryBean`에서 property 설정 정보를 참조하여 RMI 서비스에 접근하고 있다.

- **Configuration**

다음은 RMI 서비스에 접근하는 `RmiProxyFactoryBean`의 속성을 정의한 `context-remoting-rmi-client.xml`의 일부이다.

```

<!-- Add RMI Client -->
<bean
    id="movieServiceClient" class="org.springframework.remoting.rmi.RmiProxyFactoryBean">

    <property name="serviceUrl" value="rmi://localhost:1099/MovieService" />
    <property name="serviceInterface"
value="org.anyframe.sample.remoting.moviefinder.service.MovieService"/>
</bean>

```

2.Hessian

Hessian과 Burlap은 HTTP를 통해 경량의 원격 서비스를 가능하게 하는 Caucho Technology [http://www.caucho.com/]가 제공하는 솔루션이다.

- RMI의 방화벽 문제 해결
- 메모리와 저장 공간이 제한된 환경에서 사용 적합(애플릿/무선 단말기)
- 제약 사항 - 자바 표준 직렬화 매커니즘이 아닌 자체 직렬화 매커니즘 사용으로 복합 데이터 모델 불충분

Spring을 사용하지 않는 경우에도 Hessian 서비스를 작성하는 것은 매우 쉽다. 서비스 클래스가 com.caucho.hessian.server.HessianServlet을 확장하도록 하고 노출 대상 메소드의 지시자를 public으로 설정하면 된다. Spring 기반으로 Hessian 서비스를 작성하는 경우 Dependency Injection, Spring AOP 등의 Spring의 기능을 모두 이용할 수 있으므로 Spring Remoting 기능으로 제공하고 있다. RMI 서비스 작성 시 Spring 설정 파일에 RmiServiceExporter 빈을 설정한 것과 마찬가지로 Hessian 서비스 작성시에는 HessianServiceExporter 빈을 사용한다. Property 설정이 거의 유사하지만 RmiServiceExporter빈에서 설정했던 serviceName, registryPort Property 설정은 지정하지 않는다. Hessian 서비스는 RMI 레지스트리를 갖고 있지 않으므로 서비스 명과 Port 번호 설정이 필요하지 않다.

다음은 Hessian 기능을 Server와 Client 단에서 어떻게 사용해야 하는지에 대한 사용법이다.

2.1.Server Configuration

서버 구현 방식은 일반 서비스 빈 개발 방식과 같으며 HessianServiceExporter 클래스를 이용하여 손쉽게 일반 Spring Bean으로 작성된 서비스를 Hessian Service로 노출시킬 수 있다. 이때 모든 public 메소드는 서비스 메소드로 노출된다.

Property Name	Description	Required	Default Value
service	Hessian 서비스로 노출시키고 싶은 Spring Bean의 id를 설정한다.	Y	N/A
serviceInterface	Hessian 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

2.1.1.Samples

다음은 Hessian 서버 구현 속성 설정에 대한 예제이다. 서비스는 일반 Spring Bean 개발과 동일하며 HessianServiceExporter Bean에서 property 설정 정보를 참조하여 Hessian 서비스로 노출시키고 있다.

- **Configuration**

다음은 Hessian 서비스를 지원하는 HessianServiceExporter의 속성을 정의한 context-remoting-hessian.xml 의 일부이다.

```
<bean id="org.anyframe.sample.remoting.moviefinder.service.MovieService"
      class="org.anyframe.sample.remoting.moviefinder.service.impl.MovieServiceImpl">
  <property name="movieDao" ref="movieDao"/>
</bean>

<bean id="movieDao"
      class="org.anyframe.sample.remoting.moviefinder.service.impl.MovieDao" />

<!-- Add Hessian ServiceExporter -->
<bean id="hessianMovieService"
      class="org.springframework.remoting.caucho.HessianServiceExporter">
```

```

    <property name="service"
ref="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
    <property name="serviceInterface"
        value="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
</bean>

```

여기서 HessianServiceExporter 빈은 Spring MVC의 컨트롤러로 작성되어 있으므로 Spring MVC의 DispatcherServlet을 web.xml 파일에 설정해야 한다. HTTP로 서비스를 제공하기 위해서 웹 어플리케이션으로 서비스를 배포하여 제공하고 있다. 다음은 서버 사이드 웹 어플리케이션의 web.xml의 일부이다.

```

<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/context-remoting-hessian.xml</param-value>
  </context-param>
  ...중략...
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
  ...중략...
  <servlet>
    <servlet-name>remoting</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/remoting-hessian-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>remoting</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
  ...중략...
</web-app>

```

여기서 Spring MVC의 URL Mapping 기능을 사용하여 HTTP 기반의 Hessian 컨트롤러를 호출할 수 있도록 한다. 다음은 서버 사이드 웹 어플리케이션의 remoting-hessian-servlet.xml의 일부이다. * 패턴의 모든 URL에 대한 요청이 DispatcherServlet으로 전달된 후 urlMapping 정보에 의해 hessianMovieService가 호출되어 Hessian 서비스가 제공된다.

```

<bean id="urlMappingUser"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/MovieService">hessianMovieService</prop>
    </props>
  </property>
</bean>

```

2.2. Client Configuration

클라이언트는 Spring에서 제공하는 `org.springframework.remoting.caucho.HessianProxyFactoryBean` 클래스를 사용하여 Hessian Service에 접근할 수 있다.

Property Name	Description	Required	Default Value
serviceUrl	Hessian 서비스 접근 URL 정보이다. "http://" + 서버ip + ":" + port 번호 + "/" + 서비스 명 (ex.http://localhost:9002/MovieService)	Y	N/A
serviceInterface	Hessian 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

2.2.1. Samples

다음은 Hessian 클라이언트 속성 설정에 대한 예제이다. 클라이언트는 `HessianProxyFactoryBean`에서 property 설정 정보를 참조하여 Hessian 서비스에 접근하고 있다.

- **Configuration**

다음은 Hessian 서비스에 접근하는 `HessianProxyFactoryBean`의 속성을 정의한 `context-remoting-hessian-client.xml`의 일부이다.

```
<!-- Add Hessian Client -->
<bean id="movieServiceClient"
  class="org.springframework.remoting.caucho.HessianProxyFactoryBean">

  <property name="serviceUrl" value="http://localhost:9002/MovieService" />
  <property name="serviceInterface"
    value="org.anyframe.sample.remoting.moviefinder.service.MovieService"/>
</bean>
```

RMI 서비스에 접근하는 클라이언트와 마찬가지로 `serviceInterface` Property에는 서비스가 구현하는 인터페이스 클래스를 설정하고, `serviceUrl` Property에는 서비스 URL을 작성하는데 Hessian은 HTTP 기반으로 제공되므로 HTTP URL을 작성하도록 한다.

2.3. Hessian과 Burlap의 차이점

- Hessian의 경우 RMI와 같이 클라이언트와 원격 시스템의 서비스 간 통신을 할때 바이너리 데이터를 사용한다.
- Burlap의 경우 XML 기반으로 통신한다. Hessian의 바이너리 데이터에 비해 Human-readable하다. 그러나 SOAP 메시지 기반의 Remoting과 달리 Burlap의 XML 메시지는 WSDL이나 IDL과 같은 별도의 Definition Language가 없는 간단한 구조로 구성되어 있다.
- 바이너리 데이터를 사용하는 Hessian이 네트워크 환경에서의 데이터 전송 면에서 더 효율적이다.
- 그러나 메시지 가독성이 중요한 경우 혹은 Hessian 구현이 존재하지 않는 타 Language와 통신해야 하는 경우에는 Burlap을 사용해야 한다.

3. Burlap

Hessian과 Burlap은 HTTP를 통해 경량의 원격 서비스를 가능하게 하는 Caucho Technology [http://www.caucho.com/] 가 제공하는 솔루션이다.

- RMI의 방화벽 문제 해결
- 메모리와 저장 공간이 제한된 환경에서 사용 적합(애플릿/무선 단말기)
- 제약 사항 - 자바 표준 직렬화 매커니즘이 아닌 자체 직렬화 매커니즘 사용으로 복합 데이터 모델 불충분

Spring을 사용하지 않는 경우에도 Burlap 서비스를 작성하는 것은 매우 쉽다. 서비스 클래스가 com.caucho.burlap.server.BurlapServlet을 확장하도록 하고 노출 대상 메소드의 지시자를 public으로 설정하면 된다. Spring 기반으로 Burlap 서비스를 작성하는 경우 Dependency Injection, Spring AOP 등의 Spring의 기능을 모두 이용할 수 있으므로 Spring Remoting 기능으로 제공하고 있다. RMI 서비스 작성 시 Spring 설정 파일에 RmiServiceExporter 빈을 설정한 것과 마찬가지로 Burlap 서비스 작성시에는 BurlapServiceExporter 빈을 사용한다. Property 설정이 거의 유사하지만 RmiServiceExporter빈에서 설정했던 serviceName, registryPort Property 설정은 지정하지 않는다. Burlap 서비스는 RMI 레지스트리를 갖고 있지 않으므로 서비스 명과 Port 번호 설정이 필요하지 않다. 다음은 Burlap 기능을 Server와 Client 단에서 어떻게 사용해야 하는지에 대한 사용법이다.

3.1. Server Configuration

서버 구현 방식은 일반 서비스 빈 개발 방식과 같으며 BurlapServiceExporter 클래스를 이용하여 손쉽게 일반 Spring Bean으로 작성된 서비스를 Burlap Service로 노출시킬 수 있다. 이때 모든 public 메소드는 서비스 메소드로 노출된다.

Property Name	Description	Required	Default Value
service	Burlap 서비스로 노출시키고 싶은 Spring Bean의 id를 설정한다.	Y	N/A
serviceInterface	Burlap 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

3.1.1. Samples

다음은 Burlap 서버 구현 속성 설정에 대한 예제이다. 서비스는 일반 Spring Bean 개발과 동일하며 BurlapServiceExporter Bean에서 property 설정 정보를 참조하여 Burlap 서비스로 노출시키고 있다.

- **Configuration**

다음은 Burlap 서비스를 지원하는 BurlapServiceExporter의 속성을 정의한 context-remoting-burlap.xml의 일부이다.

```
<bean id="org.anyframe.sample.remoting.moviefinder.service.MovieService"
      class="org.anyframe.sample.remoting.moviefinder.service.impl.MovieServiceImpl">
  <property name="movieDao" ref="movieDao"/>
</bean>

<bean id="movieDao"
      class="org.anyframe.sample.remoting.moviefinder.service.impl.MovieDao" />

<!-- Add Burlap ServiceExporter -->
<bean id="burlapMovieService"
```

```

class="org.springframework.remoting.caucho.BurlapServiceExporter">

  <property name="service"
ref="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
  <property name="serviceInterface"
value="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
</bean>

```

여기서 BurlapServiceExporter 빈은 Spring MVC의 컨트롤러로 작성되어 있으므로 Spring MVC의 DispatcherServlet을 web.xml 파일에 설정해야 한다. HTTP로 서비스를 제공하기 위해서 웹 어플리케이션으로 서비스를 배포하여 제공하고 있다. 다음은 서버 사이드 웹 어플리케이션의 web.xml의 일부이다.

```

<web-app>
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/WEB-INF/context-remoting-burlap.xml</param-value>
  </context-param>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>remoting</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>WEB-INF/context-remoting-burlap.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>remoting</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
</web-app>

```

여기서 Spring MVC의 URL Mapping 기능을 사용하여 HTTP 기반의 Burlap 컨트롤러를 호출할 수 있도록 한다. 다음은 서버 사이드 웹 어플리케이션의 remoting-burlap-servlet.xml의 일부이다. * 패턴의 모든 URL에 대한 요청이 DispatcherServlet으로 전달된 후 urlMapping 정보에 의해 burlapMovieService가 호출되어 Burlap 서비스가 제공된다.

```

<bean id="urlMappingUser"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/MovieService">burlapMovieService</prop>
    </props>
  </property>
</bean>

```

3.2.Client Configuration

클라이언트는 Spring에서 제공하는 org.springframework.remoting.caucho.BurlapProxyFactoryBean 클래스를 사용하여 Burlap Service에 접근할 수 있다.

Property Name	Description	Required	Default Value
serviceUrl	Burlap 서비스 접근 URL 정보이다. "http://" + 서버ip + ":" + port 번호 + "/" + 서비스 명 (ex.http://localhost:9002/MovieService)	Y	N/A
serviceInterface	Burlap 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

3.2.1.Samples

다음은 Burlap 클라이언트 속성 설정에 대한 예제이다. 클라이언트는 BurlapProxyFactoryBean에서 property 설정 정보를 참조하여 Burlap 서비스에 접근하고 있다.

- **Configuration**

다음은 Burlap 서비스에 접근하는 BurlapProxyFactoryBean의 속성을 정의한 context-remoting-burlap-client.xml 의 일부이다.

```
<!-- Add Burlap Client -->
<bean id="movieServiceClient"
  class="org.springframework.remoting.caucho.BurlapProxyFactoryBean">

  <property name="serviceUrl" value="http://localhost:9002/MovieService" />
  <property name="serviceInterface"
    value="org.anyframe.sample.remoting.moviefinder.service.MovieService"/>
</bean>
```

RMI 서비스에 접근하는 클라이언트와 마찬가지로 serviceInterface Property에는 서비스가 구현하는 인터페이스 클래스를 설정하고. serviceUrl Property에는 서비스 URL을 작성하는데 Burlap은 HTTP 기반으로 제공되므로 HTTP URL을 작성하도록 한다.

3.3.Hessian과 Burlap의 차이점

- Hessian의 경우 RMI와 같이 클라이언트와 원격 시스템의 서비스 간 통신을 할때 바이너리 데이터를 사용한다.
- Burlap의 경우 XML 기반으로 통신한다. Hessian의 바이너리 데이터에 비해 Human-readable하다. 그러나 SOAP 메시지 기반의 Remoting과 달리 Burlap의 XML 메시지는 WSDL이나 IDL과 같은 별도의 Definition Language가 없는 간단한 구조로 구성되어 있다.
- 바이너리 데이터를 사용하는 Hessian이 네트워크 환경에서의 데이터 전송 면에서 더 효율적이다.
- 그러나 메시지 가독성이 중요한 경우 혹은 Hessian 구현이 존재하지 않는 타 Language와 통신해야 하는 경우에는 Burlap을 사용해야 한다.

4.HTTP Invoker

HTTP Invoker는 HTTP를 통해 경량의 원격 서비스를 가능하게 하는 Spring에서 제공하는 Remoting 기능이다. 앞서 소개한 RMI와 Hessian/Burlap의 경우 아래와 같은 단점이 있는데 이러한 단점을 보완해 주는 기능을 HTTP Invoker가 가지고 있다.

- RMI 단점: RMI의 경우 자바의 표준 객체 직렬화를 사용하지만 방화벽을 통과하기가 어렵다.
- Hessian/Burlap의 단점: 방화벽을 쉽게 통과하지만, 자체적인 객체 직렬화 매커니즘을 사용하여 복잡한 형태의 객체 직렬화 시 문제가 발생할 수 있다.

HTTP Invoker는 RMI와 Hessian/Burlap의 단점을 보완해준다. Spring에서 제공하는 Remoting 기능으로 HTTP를 통해 Remoting 기능을 수행하며 자바의 표준 직렬화 매커니즘을 사용한다. 단, **Spring**에서만 제공하는 **Remoting** 기술로 클라이언트와 원격 서비스 모두 **Spring** 어플리케이션 이어야 한다.

다음은 HTTP Invoker 기능을 Server와 Client 단에서 어떻게 사용해야 하는지에 대한 사용법이다.

4.1.Server Configuration

서버 구현 방식은 일반 서비스 빈 개발 방식과 같으며 HttpInvokerServiceExporter 클래스를 이용하여 손쉽게 일반 Spring Bean으로 작성된 서비스를 HTTP Invoker Service로 노출시킬 수 있다. 이때 모든 public 메소드는 서비스 메소드로 노출된다.

Property Name	Description	Required	Default Value
service	HTTP Invoker 서비스로 노출시키고 싶은 Spring Bean의 id를 설정한다.	Y	N/A
serviceInterface	HTTP Invoker 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

4.1.1.Samples

다음은 HTTP Invoker 서버 구현 속성 설정에 대한 예제이다. 서비스는 일반 Spring Bean 개발과 동일하며 HttpInvokerServiceExporter Bean에서 property 설정 정보를 참조하여 HTTP Invoker 서비스로 노출시키고 있다.

- **Configuration**

다음은 HTTP Invoker 서비스를 지원하는 HttpInvokerServiceExporter의 속성을 정의한 context-remoting-httpinvoker.xml 의 일부이다.

```
<bean id="httpinvokerMovieService"
      class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
  <property name="service"
    ref="org.anyframe.sample.remoting.moviefinder.service.MovieService"/>
  <property name="serviceInterface"
    value="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
</bean>
```

여기서 HttpInvokerServiceExporter 빈은 Spring MVC의 컨트롤러로 작성되어 있으므로 Spring MVC의 DispatcherServlet을 web.xml 파일에 설정해야 한다. HTTP로 서비스를 제공하기 위해서 웹 어플리케이션으로 서비스를 배포하여 제공하고 있다. 다음은 서버 사이드 웹 어플리케이션의 web.xml의 일부이다.

```
<web-app id="webApp_ID" version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>sample-web</display-name>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>WEB-INF/context-remoting-httpinvoker.xml</param-value>
</context-param>

...중략...

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>

<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/remoting-httpinvoker-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>

<!-- remoting-configuration-START -->
<servlet>
    <servlet-name>remoting</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:/springmvc/remoting-servlet.xml</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>remoting</servlet-name>
    <url-pattern>/remoting/*</url-pattern>
</servlet-mapping>
<!-- remoting-configuration-END -->
...중략...
</web-app>

```

여기서 Spring MVC의 URL Mapping 기능을 사용하여 HTTP 기반의 HTTP Invoker 컨트롤러를 호출할 수 있도록 한다. 다음은 서버 사이드 웹 어플리케이션의 remoting-httpinvoker-servlet.xml 의 일부이다. * 패턴의 모든 URL에 대한 요청이 DispatcherServlet으로 전달된 후 urlMapping 정보에 의해 remotingMovieService가 호출되어 HTTP Invoker 서비스가 제공된다.

```

<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/MovieService">httpinvokerMovieService</prop>
        </props>
    </property>

```

```
</bean>
```

4.2.Client Configuration

클라이언트는 Spring에서 제공하는 `org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean` 클래스를 사용하여 HTTP Invoker Service에 접근할 수 있다.

Property Name	Description	Required	Default Value
serviceUrl	HTTP Invoker 서비스 접근 URL 정보이다. "http://" + 서버ip + ":" + port 번호 + "/" + 서비스 명 (ex.http://localhost:9002/MovieService)	Y	N/A
serviceInterface	HTTP Invoker 서비스로 노출되는 서비스의 인터페이스 클래스를 패키지정보와 함께 작성한다.	Y	N/A

4.2.1.Samples

다음은 HTTP Invoker 클라이언트 속성 설정에 대한 예제이다. 클라이언트는 `HttpInvokerProxyFactoryBean`에서 property 설정 정보를 참조하여 HTTP Invoker 서비스에 접근하고 있다.

- **Configuration**

다음은 HTTP Invoker 서비스에 접근하는 `HttpInvokerProxyFactoryBean`의 속성을 정의한 `context-remoting-httpinvoker-client.xml` 의 일부이다.

```
<bean id="movieServiceClient"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
  <property name="serviceUrl"
    value="http://localhost:9002/MovieService" />
  <property name="serviceInterface"
    value="org.anyframe.sample.remoting.moviefinder.service.MovieService" />
</bean>
```

Hessian/Burlap 서비스에 접근하는 클라이언트와 마찬가지로 `serviceInterface` Property에는 서비스가 구현하는 인터페이스 클래스를 설정하고, `serviceUrl` Property에는 서비스 URL을 작성하는데 HTTP Invoker라는 이름에서도 알 수 있듯이 HTTP 기반으로 제공되므로 HTTP URL을 작성하도록 한다. HTTP 방식으로 원격 서비스를 호출할 때 HTTP 클라이언트를 선택할 수 있다. 기본적으로 `HttpInvokerProxyFactoryBean`은 J2SE HTTP 클라이언트를 사용하지만, `httpInvokerRequestExecutor` 프라퍼티를 셋팅하여 Apache Commons `HttpClient`를 사용할 수도 있다.

```
<!-- Add HTTP Invoker Client -->
<bean id="movieServiceClient"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
  <property name="serviceUrl" value="http://localhost:9002/MovieService" />
  <property name="serviceInterface"
    value="org.anyframe.sample.remoting.moviefinder.service.MovieService"/>
  <property name="httpInvokerRequestExecutor" ref="httpClient"/>
</bean>

<bean id="httpClient"
  class="org.springframework.remoting.httpinvoker.CommonsHttpInvokerRequestExecutor">
  <property name="httpClient">
    <bean class="org.apache.commons.httpclient.HttpClient">
      <property name="connectionTimeout" value="2000"/>
    </bean>
  </property>
```

```
</bean>
```

- **Test Case**

다음은 앞서 정의한 속성 설정 파일들을 기반으로 하여 HTTP Invoker 서비스에 접근하는 예제인 MovieController.java 코드의 일부이다.

```
@Controller
@RequestMapping("/remotingMovieExporter.do")
public class MovieController {
    @Inject
    @Named("remotingMovieClient")
    private MovieService movieService;

    @RequestMapping(params = "method=get")
    public String get(@RequestParam("movieId") String movieId, Model model)
        throws Exception {
        Movie movie = this.movieService.get(movieId);
        if (movie == null) {
            throw new Exception("Resource not found " + movieId);
        }
        model.addAttribute(movie);

        return "remoting/moviefinder/movie/form";
    }

    // ...
}
```

5.Resources

- 다운로드

다음에서 sample 코드를 포함하고 있는 anyframe-sample-remoting.zip 파일을 다운받은 후, 압축을 해제한다.

- Maven 기반 실행

Command 창에서 압축 해제 폴더로 이동한 후, mvn clean test 라는 명령어를 실행시켜 결과를 확인한다.

- Eclipse 기반 실행

Eclipse에서 압축 해제 프로젝트를 import한 후, src/test/java 폴더의 모든 Test 코드 각각에 대해 마우스 오른쪽 버튼을 클릭하고, 컨텍스트 메뉴에서 Run As > JUnit Test를 클릭한다. 그리고 실행 결과를 확인한다.

표 5.1. Download List

Name	Download
anyframe-sample-remoting.zip	Download [http://dev.anyframejava.org/docs/anyframe/plugin/optional/remoting/1.0.2/reference/sample/anyframe-sample-remoting.zip]