

Nell'articolo di oggi vedremo le API che Spring-Ws mette a disposizione per la realizzazione di un client.

Alla base di tutto vi è la classe `WebServiceTemplate` che offre i metodi per l'invio e la ricezione dei messaggi. I messaggi possono essere inviati e ricevuti come puro stream oppure possono essere soggetti al processo di marshalling e unmarshalling.

Occorre prima di tutto definire il nostro client nei file di configurazione di Spring, tramite la solita definizione di bean.

```
<bean id="client" class="org.springframework.ws.client.core.WebServiceTemplate"><property name="defaultUri" value="http://example.com/WebService" /> <property name="messageFactory" ref="messageFactory" /> </bean>
```

La property `defaultUri` indica la URL dei servizi da invocare, mentre la `messageFactory` individua la classe responsabile della generazione dei messaggi da inviare.

Come factory Spring mette a disposizione 2 implementazioni:

- `SaajSoapMessageFactory` basata su SAAJ
- `AxiomSoapMessageFactory` basata su Axiom

In assenza del valore Spring userà di default l'implementazione basata su SAAJ.

Nei file xml di Spring-Ws setteremo la factory desiderata come

```
<bean id="messageFactory" class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory" />
<bean id="messageFactory" class="org.springframework.ws.soap.axiom.AxiomSoapMessageFactory" />
```

A questo possiamo sfruttare il nostro client per invocare il servizio e gestirne il risultato.

```
StreamSource source = new StreamSource(new StringReader(MESSAGE));
```

```
StreamResult result = new StreamResult(System.out);
```

```
client.sendSourceAndReceiveToResult(source, result);
```

In source è disponibile il messaggio nel formato atteso dal servizio remoto e nella System.out verrà riversato la risposta del server.

Invocazione conclusa con minimo sforzo. A questo punto andrebbe analizzata la risposta per gestire le nuove azioni da intraprendere.

Come ulteriore aiuto Spring-Ws ci dà la possibilità di sfruttare le tecniche di marshalling e unmarshalling per ottenere degli oggetti

già pronti per l'uso.

Per fare ciò occorre valorizzare il marshaller e l'unmarshaller all'interno del nostro client.

```
<bean id="client" class="org.springframework.ws.client.core.WebServiceTemplate">
```

```
<property name="defaultUri" value="http://example.com/WebService" />
```

```
<property name="messageFactory" ref="messageFactory" />
```

```
<property name="marshaller" ref="Marshaller" />
```

```
<property name="unmarshaller" ref="Marshaller" />
```

```
</bean>
```

Per la definizione del marshaller si rimanda all' [articolo](#) dedicato.

Definito il marshaller e l'unmarshaller verrà usato il seguente metodo che restituisce un oggetto valorizzato

```
Object response = client.marshalSendAndReceive(requestPayload);
```

dove requestPayload è il bean che descrive la richiesta del sistema opportunamente valorizzata.

Nel caso in cui dobbiate gestire gli header SOAP dovete definire una classe che implementi l'interfaccia `WebServiceMessageCallback`.

Tale interfaccia definisce il metodo `public void doWithMessage(WebServiceMessage message)` che consente di

modificare il messaggio prima del suo invio al server.

In questo caso invocherete il metodo

```
client.marshallSendAndReceive(requestPayload, new WebServiceMessageCallback() { public  
void doWithMessage(WebServiceMessage message) {  
((SoapMessage)message).setSoapAction("action");  
}  
});
```

[Prossimo](#)