**Web Services III: SOAP Handlers**

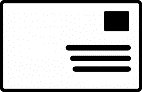
**Objectivos**

* Construir e modificar mensagens SOAP
* Desenvolver *SOAP Handlers* para modificar as mensagens XML do Web Service

**Índice:**

* [SOAP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/index.html#soap)
* [Handlers](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/index.html#handlers)
  + [Configuração](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/index.html#handlers-config)
  + [Biblioteca ws-handlers](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/index.html#bib-handlers)
  + [Exemplo de inspeção de mensagens](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/index.html#ex-handlers)

**SOAP**



O SOAP é o formato de mensagens para Web Services. Os envelopes podem ser transportados pela rede de diversas formas, mas a mais comum é através do protocolo HTTP. O SOAP é independente do protocolo de transporte.

Os documentos seguintes são mensagens SOAP correspondentes a um par pedido-resposta.

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:ns1="urn:hello">

<soapenv:Body>

<ns1:sayHello>

<ns1:name>friend</ns1:name>

</ns1:sayHello>

</soapenv:Body>

</soapenv:Envelope>

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:ns1="urn:hello">

<soapenv:Body>

<ns1:sayHelloResponse>

<ns1:return>Hello friend!</ns1:return>

</ns1:sayHelloResponse>

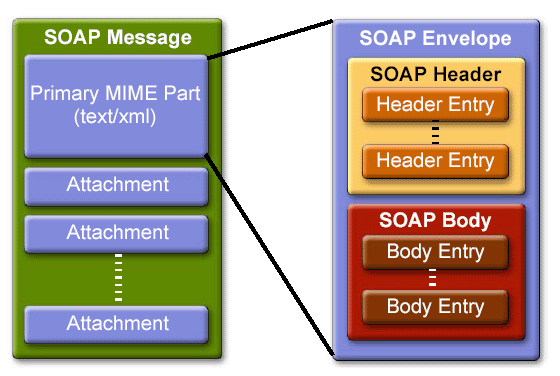
</soapenv:Body>

</soapenv:Envelope>

Uma mensagem SOAP é um documento XML designado por *envelope*. O cabeçalho *header* permite a composição de protocolos, pois cada elemento de extensão indica a sua versão e a opcionalidade da sua interpretação.  
O corpo (*body*) contém os dados de negócio da mensagem ou então o elemento *Fault* com informação de erro.

A SAAJ (SOAP with Attachments API for Java) é uma biblioteca que estende o XML DOM, adaptando-o para documentos XML que são **mensagens SOAP**. Isto significa que existem vários métodos específicos para tratar as mensagens.

Na biblioteca SAAJ, uma mensagem SOAP tem a seguinte estrutura:



A SOAPMessage contém várias partes. A primeira parte é uma SOAPPart, que contém um SOAPEnvelope.  
Um SOAPEnvelope contém um SOAPBody e opcionalmente um SOAPHeader.  
Dentro destes, podem ser colocados SOAPElement.  
Numa mensagem SOAP, os elementos devem ser sempre especificados com espaço de nomes, para evitar conflitos.

Os objectos da biblioteca SAAJ estão no pacote javax.xml.soap.\*  
O exemplo seguinte mostra como se constrói uma mensagem simples.

...

MessageFactory mf = MessageFactory.newInstance();

SOAPMessage soapMessage = mf.createMessage();

SOAPPart soapPart = soapMessage.getSOAPPart();

SOAPEnvelope soapEnvelope = soapPart.getEnvelope();

SOAPBody soapBody = soapEnvelope.getBody();

Name name = soapEnvelope.createName("HelloWorld", "hw", "urn:helloworld");

SOAPElement element = soapBody.addChildElement(name);

element.addTextNode( "hello text message" );

...

soapMessage.writeTo(System.out);

System.out.println();

...

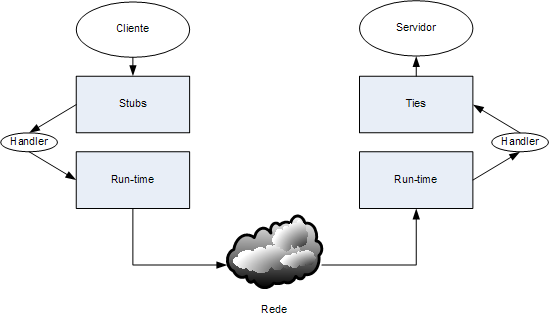
**Exemplos SOAP e XML**

Os exemplos seguintes mostram como modificar as mensagens SOAP e como enviar mensagem SOAP directamente sem *stubs*.  
Para complementar existem também exemplos das tecnologias base de XML: SAX, DOM, XSD, XPath.

* [SOAP [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/soap.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/soap.zip) - SOAP API for JAVA (SAAJ)
* [XML [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/xml.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/xml.zip) - Processamento de XML em JAVA

**SOAP Handlers**

A biblioteca de Web Services para Java, JAX-WS, tem um mecanismo que permite intercetar e aceder diretamente às mensagens SOAP.



Um *Handler* é uma classe Java que implementa a interface javax.xml.ws.handler.soap.SOAPHandler.

O seguinte exemplo é um *Handler* básico. O principal método é o handleMessage() que é invocado de cada vez que chega ou parte uma mensagem:

package example.ws.handler;

import java.util.Set;

import javax.xml.namespace.QName;

import javax.xml.ws.handler.MessageContext;

import javax.xml.ws.handler.soap.SOAPHandler;

import javax.xml.ws.handler.soap.SOAPMessageContext;

public class EmptyHandler implements SOAPHandler {

/\*\*

\* Gets the names of the header blocks that can be processed by this Handler instance.

\* If null, processes all.

\*/

public Set getHeaders() {

return null;

}

/\*\*

\* The handleMessage method is invoked for normal processing of inbound and

\* outbound messages.

\*/

public boolean **handleMessage**(SOAPMessageContext smc) {

return true;

}

/\*\* The handleFault method is invoked for fault message processing. \*/

public boolean handleFault(SOAPMessageContext smc) {

return true;

}

/\*\*

\* Called at the conclusion of a message exchange pattern just prior to the

\* JAX-WS runtime dispatching a message, fault or exception.

\*/

public void close(MessageContext messageContext) {

}

}

**Retorno e Exceções**

O retorno do método handleMessage() determina de que forma prossegue o processamento da mensagem.  
Se for 'true' o processamento deve prosseguir; se for 'false' bloqueia o processamento da mensagem, mudando-lhe o sentido e fazendo-a voltar para o cliente.

A utilização de **exceções** permite modificar o normal processamento das mensagens SOAP:

* Se um *handler* atirar uma exceção java.lang.RuntimeException (ou um sub-tipo desta classe), indica que o processamento normal da mensagem deve parar. O método close() é invocado nos *handlers* que entretanto já tinham sido chamados, o sentido da mensagem é invertido e a excepção é despachada. Se a mensagem tinha o sentido de saída (*outbound*), é gerada uma mensagem SOAP automaticamente preenchida com uma SOAP *Fault*.
* Se um *handler* atirar uma exceção javax.xml.ws.soap.SOAPFaultException (que é um sub-tipo de javax.xml.ws.ProtocolException), indica que o processamento normal da mensagem deve parar e que se deve iniciar o processamento de mensagem de erro. O sentido da mensagem é invertido, se a mensagem não é ainda uma mensagem de Fault é substituída por uma mensagem de Fault, e o método handleFault() é invocado para o próximo *Handler* caso exista, para que seja preenchida manualmente a informação da *Fault*.

**Configuração**

A configuração dos *Handlers* é efectuada num ficheiro que define a cadeia de *handlers*.

<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">

<handler-chain>

<handler>

<handler-class>example.ws.handler.LoggingHandler</handler-class>

</handler>

</handler-chain>

</handler-chains>

A configuração os *Handlers* é diferente no servidor e no cliente. Em ambos os casos não são necessárias alterações aos pom.xml.

* No **servidor**:
  + Colocar o ficheiro de configuração handler-chain.xml na pasta src/main/resources
  + Acrescentar a seguinte anotação na classe de implementação: @HandlerChain(file="/handler-chain.xml")
* No **cliente**:
  + Colocar o ficheiro de configuração handler-chain-binding.xml na pasta src/jaxws (definida por convenção)

**Biblioteca ws-handlers**

Esta biblioteca é um módulo Maven com *Handlers* que podem ser usados em servidores e clientes.

* [Biblioteca de JAX-WS Handlers [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/ws-handlers.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/ws-handlers.zip)  
  - contém:
  + LoggingHandler - imprime mensagens SOAP para System.out
  + MessageContextHandler - imprime variáveis de contexto para System.out
  + HeaderHandler - acrescenta um Header à mensagem SOAP
  + HeaderHandlerTest - usa *mocks* para testar localmente um handler

- para usar:

* + fazer mvn install para compilar e instalar o módulo no repositório local do Maven
  + acrescentar a dependência para example:ws-handlers:1.0-SNAPSHOT no pom.xml do projeto que vai usar os *Handlers*
  + editar/acrescentar o ficheiro de configuração dos *handlers* no projeto que os vai usar

Esta biblioteca poderá ser enriquecida com *handlers* adicionais (por exemplo, no projeto).

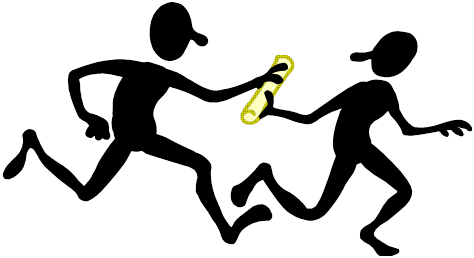
**Exemplo de *Handlers* de inspeção de mensagens**



Este exemplo demonstra a forma como os SOAP Handlers acedem às mensagens XML dos Web Services.

* [Web Service com JAX-WS Handlers [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws_handlers.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws_handlers.zip)  
  - os SOAP Handlers permitem intercetar as mensagens SOAP que são enviadas e recebidas.  
  A configuração é feita através de um ficheiro e de uma anotação.
* [Web Service client com JAX-WS Handlers [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws-cli_handlers.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws-cli_handlers.zip)  
  - a definição de handlers no cliente é feita através de um ficheiro de configuração.

**Exemplo de *Handlers* estafetas de dados**



Este exemplo demonstra os mecanismos de passagem de dados entre as diversas camadas de um Web Service:

1. do cliente para o *handler* cliente (via contexto do pedido, que é obtido a partir do *stub*),
2. do *handler* cliente para o *handler* servidor (via cabeçalho na mensagem SOAP),
3. do *handler* servidor para o servidor (via contexto das mensagens que é fornecido como argumento),
4. do servidor para o *handler* servidor (novamente via contexto das mensagens que é fornecido pela biblioteca através da anotação @Resource),
5. do *handler* servidor para o *handler* cliente (via cabeçalho na mensagem SOAP),
6. e finalmente, do *handler* cliente para o servidor (via contexto da resposta).

Consultar os exemplos seguindo os comentários numerados #1, #2, #3, ... que seguem a sequência de uma invocação remota começando no cliente, passando pelo servidor e voltando ao cliente. Pelo caminho o *token* vai sendo acrescentado com mais dados.

* [Web Service Relay [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws_handlers_relay.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws_handlers_relay.zip)
* [Web Service client Relay [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws-cli_handlers_relay.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws-cli_handlers_relay.zip)

**Exercício**

**Terceira parte do projeto**

O objetivo deste exercício é configurar o *LoggingHandler* para intercetar as comunicações.

Começar por supplier-ws:

1. Adicionar a dependência para a biblioteca ws-handlers no pom.xml.
2. Na pasta src/main/resources adicionar ficheiro supplier-ws\_handler-chain.xml com configuração da cadeia de interceção.  
   O *LoggingHandler* deve ser chamado no início e no fim da cadeia.
3. Adicionar a seguinte anotação ao PortImpl:

@HandlerChain(file = "/supplier-ws\_handler-chain.xml")

1. Lançar o servidor.
   * cd supplier-ws
   * mvn compile exec:java
2. Usar o cliente para chamar o ping.
   * cd supplier-ws-cli
   * mvn compile exec:java
3. Agora, ao receber mensagens, estas devem ser capturadas e impressas para a consola do servidor.

Configurar os *handlers* supplier-ws-cli:

1. Adicionar também a dependência para a biblioteca ws-handlers.
2. Criar os ficheiros de configuração da cadeia de handlers na pasta src/jaxws
   * Para ver ficheiros de configuração semelhantes, consultar este [exemplo](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/hello-ws-cli_handlers.zip).
3. Para verificar se a configuração ficou bem feita, fazer uma chamada ao ping e confirmar que as mensagens são também capturadas e impressas na consola do lado do cliente.

Neste ponto as mensagens SOAP estão a ser capturadas à saída no cliente e à chegada no servidor.

Vamos agora criar um novo *handler* para validação temporal das mensagens:

1. Criar um *handler* que acrescenta um cabeçalho simples à saída do cliente (*outbound message*) com a data e hora atual.
2. O novo *handler* deve ler o cabeçalho à chegada ao servidor (*inbound message*).
3. Confirmar que o novo *handler* funciona como esperado.  
   O *LoggingHandler* deve ser chamado no início e no fim da cadeia para ver a mensagem antes e depois da modificação.
4. Modificar o *handler* para aplicar a seguinte regra:  
   Caso a diferença temporal seja maior do que 3 segundos, rejeitar a mensagem  
   (ver secção sobre [retorno e exceções](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/08-ws3/index.html#handlers-ex) em *handlers*)
5. De que forma se poderá testar a rejeição de mensagem?

Próximos passos:

* Para cada requisito de segurança referido no enunciado do projeto:
  + Identificar que alterações são necessárias às mensagens SOAP.
    - O que se deve fazer à saída da mensagem?
    - O que se deve fazer à chegada?
      * O que fazer quando a mensagem não corresponde ao esperado?
  + Definir *handlers* (e outros objetos) para implementar as alterações necessárias
  + Implementar de forma gradual.

**Bom trabalho!**