Frameworks pour SOAP aujourd’hui

Frameworks

* Beaucoup de framework ou API cassés depuis Java 1.9

* Apache Axis 2
  * Framework Rest et Soap hypercomplet
  * Ligne de cmd
  * Version C++
  * Gen. De code
  * Mais <= Java 1.8 ☹

L'arbre qui cache la foret?

Axis with attachement

http://axis.apache.org/axis2/java/core/docs/mtom-guide.html#a3

Axis Jaxws (il fait tout ?)

http://axis.apache.org/axis2/java/core/docs/jaxws-guide.html

Axis security module : Rampart (WS-Security)

http://axis.apache.org/axis2/java/rampart/

Axis reliable messaging (ACK)

http://axis.apache.org/axis2/java/sandesha/index.html

Axis C

http://axis.apache.org/axis2/c/core/

Frameworks

* Beaucoup de framework ou API cassés depuis Java 1.9
* Apache CFX ?
  * A jour Rest et Soap (connaît pas, à l'air compliqué)
* JAX-WS (Oracle, metro)
  * Voir après
  * Gen. De code, pas très complexe

Frameworks

* Beaucoup de framework ou API cassés depuis Java 1.9

* SpringBoot
  * Faisable
  * Code client ok
  * Code serveur (schema XML à faire à la main) ☹

Java API for XML Web Services (JAX-WS)

JAXWS

* API standard pour créer des services et des clients

* Doc. D'une implémentation : https://javaee.github.io/metro-jax-ws/doc/user-guide/release-documentation.html

JAXWS

Outils

* https://javaee.github.io/metro-jax-ws/doc/user-guide/ch04.html
* **Wsimport : gen. De services, de clients**
* **Wsgen : gen WSDL, schémas, fichiers de déploiement de services**

JAXWS

* API standard pour créer des services et des clients
* Doc. D'une implémentation : https://javaee.github.io/metro-jax-ws/doc/user-guide/release-documentation.html

* Création de services
  * 2 approches
    * **Top-Down -> depuis le WSDL**
    * **Bottom-Up-> depuis un POJO**

JAXWS

* API standard pour créer des services et des clients
* Création de services
  * **Top-Down -> depuis le WSDL**

* Génération de code avec *wsimport -s . -p package URI-du-WSDL*

* *Je ne vois personne écrire un fichier WSDL ??? Et vous?*

JAXWS

* API standard pour créer des services et des clients
* Création de services
  * 2 approches
    * **Bottom-Up-> depuis un POJO**

  * Annotation d'un POJO et génération du WSDL
  * Plus simple

JAXWS

* **Bottom-Up-> depuis un POJO**
  * Exemple classique

```
public class Employee {
   private int id;
   private String firstName;

   // standard getters and setters
}
```

Bien ajouter les getters et setters et les constructeurs, ne pas oublier le const. Sans paramètres

JAXWS

* **Bottom-Up-> depuis un POJO**
  * Exemple classique

```
@WebService
public interface EmployeeService {
  @WebMethod
  Employee getEmployee(int id);

  @WebMethod
  Employee updateEmployee(int id, String name);

  @WebMethod
  Employee updateEmployee(int id, String name);

  @WebMethod
  boolean deleteEmployee(int id);

  @WebMethod
  Employee addEmployee(int id, String name);

  // ...
}
```

JAXWS

* **Bottom-Up-> depuis un POJO**
* **https://docs.oracle.com/cd/E13222_01/wls/docs103/webserv_ref/annotations.html**

* *@WebService* décrit une interface de service
* *@WebMethod* décrit une méthode
* *@WebResult* décrit le type et serialisation du résultat

JAXWS

* **Bottom-Up-> depuis un POJO**
* *@WebService* décrit une interface de service

```
@WebService(serviceName="ComplexService",
name="ComplexPortType",
targetNamespace="http://example.org")
```

JAXWS

* **Bottom-Up-> depuis un POJO**
* *@WebMethod* décrit une méthode

```
@WebMethod(operationName="echoComplexType",
targetNamespace="http://example.org")
```

JAXWS

* **Bottom-Up-> depuis un POJO**
* @WebParam *@WebResult* décrivent les type et serialisation des paramètres et résultats

```
@WebMethod()
@WebResult(name="IntegerOutput",
targetNamespace="http://example.org/complex")

public int echoInt(
@WebParam(name="IntegerInput",
targetNamespace="http://example.org/complex")
int input)
```

JAXWS

* @WebMethod(exclude=true) exclue une opération

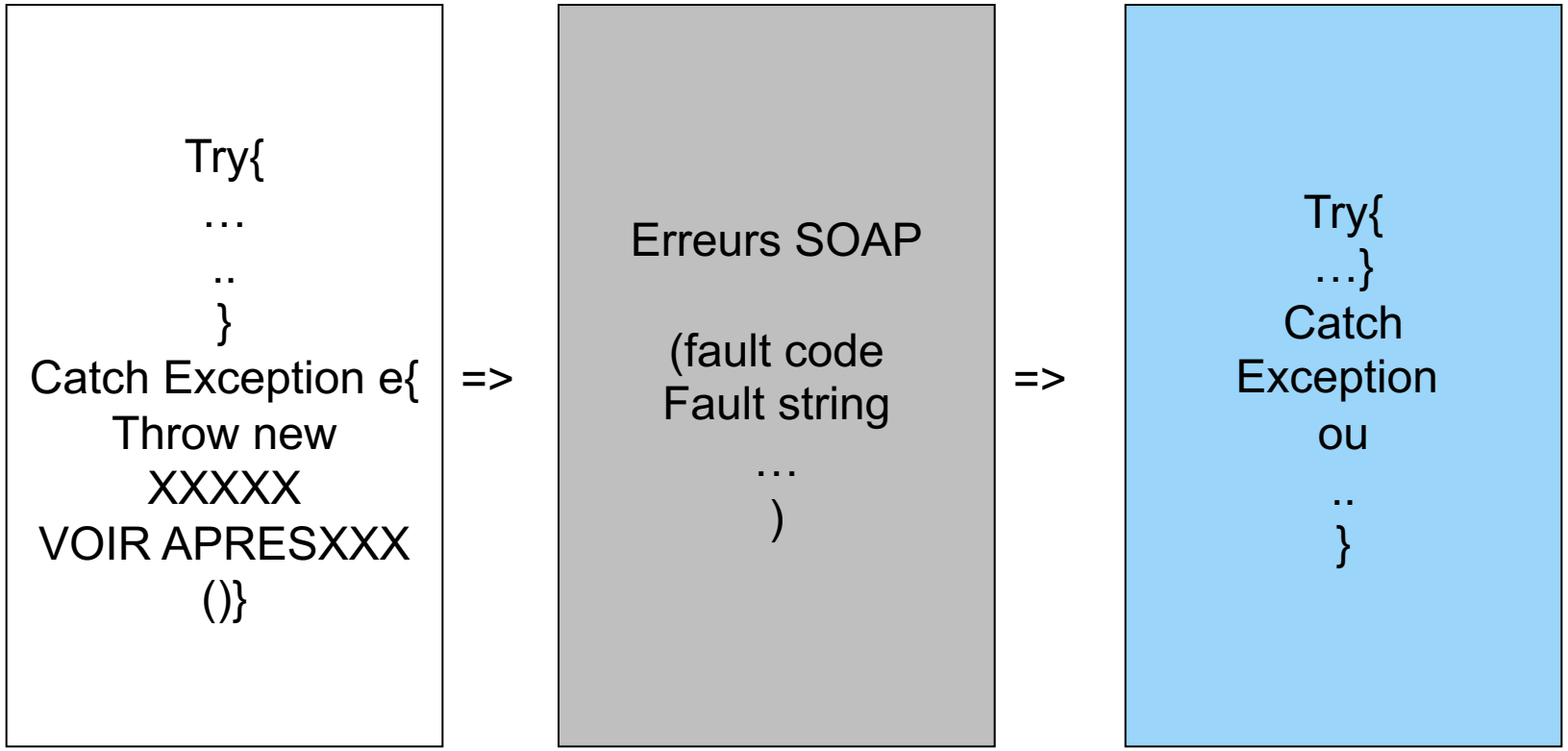
* Mais aussi @OneWay,

Means That an operation not return a value to the calling application.

JAWS et Gest. Des Exceptions, Stateful WS

Gestion d'erreurs

1. Gestion des exceptions BIG PICTURE



Le message texte est encapsulé dans fault string et est récupérable chez le client

JAXWS et gest. Des exceptions

MODE FACILE

Utiliser **Unmodeled Faults =>** Runtime exception

throw new RuntimeException("Please enter a name."); //Unmodeled fault

throw new WebServiceException(("Please enter a name.");
public WebServiceException(String message, Throwable cause)

JAXWS et gest. Des exceptions

MODE PERSONALISE => **Modeled Faults**
https://docs.oracle.com/cd/E24329_01/web.1211/e24965/faults.htm#WSADV633

Exemple complet avec FaultString et Detail :

Exception CheckVerifyFault

JAXWS et gen. De SoapFault

```
@WebFault(name="CheckVerifyFault")
public class CheckVerifyFault extends Exception {

private CheckFaultBean faultInfo;

    public CheckVerifyFault(String message, CheckFaultBean faultInfo) {
        super(message);
        this.faultInfo = faultInfo;
    }

    public CheckVerifyFault(String message, CheckFaultBean faultInfo,
            Throwable cause) {
        super(message, cause);
        this.faultInfo = faultInfo;
    }

    public CheckFaultBean getFaultInfo() {
        return faultInfo;
    }
}
```

JAXWS et gen. De SoapFault

```
class CheckFaultBean {

private String message;

    public CheckFaultBean() {
    }
    public CheckFaultBean(String message) {
        this.message = message;
    }

    public String getMessage() {
        return message;
    }

}
```

JAXWS et gen. De SoapFault

Dans le code du service:

throw new CheckVerifyFault("Fault Detected", new CheckFaultBean("detail"));

Dans le WSDL généré :

```
<operation name="getbook"><input wsam:Action="http://ws/S1/getbookRequest"
message="tns:getbook"/><output wsam:Action="http://ws/S1/getbookResponse"
message="tns:getbookResponse"/>

<fault message="tns:CheckVerifyFault" name="CheckVerifyFault"
wsam:Action="http://ws/S1/getbook/Fault/CheckVerifyFault"/>

</operation>
```

Génération de code client

JAXWS

* Génération de clients
* *wsimport -keep -p packageclient URI-WSDL*

* Génération de stubs qui masquent la sérialisation (avec JAXB) et la désérialisation des paramètres et résultats
* Il suffit d'appeler le stub correctement pour faire un client

JAXWS

* Génération de clients

```
public class EmployeeServiceClient {
  public static void main(String[] args) throws Exception {
    URL url = new URL("http://localhost:8080/employeeservice?wsdl");

    EmployeeService_Service employeeService_Service
      = new EmployeeService_Service(url);
    EmployeeService employeeServiceProxy
      = employeeService_Service.getEmployeeServiceImplPort();

    List<Employee> allEmployees
      = employeeServiceProxy.getAllEmployees();
  }
}
```

JAXWS

* Génération de clients

Concernant Le code précédent, ce n'est pas magique, il faut interpréter le code généré et savoir l'utiliser.

Donc, le code change à chaque fois

JAXWS

* Génération de clients

Concernant Le code précédent, ce n'est pas magique, il faut interpréter le code généré et savoir l'utiliser.

Donc, le code change à chaque fois

Prenons ce service comme exemple:

```
@WebService(serviceName = "NewWebService")
public class NewWebService {

@WebMethod(operationName = "hello")
  public String hello(@WebParam(name = "name") String txt) {
    return "Hello " + txt + " !";
  }
}
```

JAXWS

* Génération de clients

Exemple :


```
@WebService(name = "NewWebService",
targetNamespace = "http://ws/")
@XmlSeeAlso({
    ObjectFactory.class
})
public interface NewWebService {


@WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "hello",
targetNamespace = "http://ws/", className = "cl2.Hello")
    @ResponseWrapper(localName = "helloResponse",
targetNamespace = "http://ws/", className =
"cl2.HelloResponse")
    public String hello(
        @WebParam(name = "name", targetNamespace = "")
        String name);

}
```

JAXWS

* Génération de clients

Exemple :


```
public class HelloResponse {

    @XmlElement(name = "return")
    protected String _return;

    /**
     * Obtient la valeur de la propriété return.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getReturn() {
        return _return;
    }
```

JAXWS

* Génération de clients

Exemple de client:

```
public static void main(String[] args) {
    // TODO code application logic here
    System.out.println(hello_1("toto"));
  }

  private static String hello_1(java.lang.String name) {
    cl2.NewWebService_Service service = new cl2.NewWebService_Service();
    cl2.NewWebService port = service.getNewWebServicePort();
    return port.hello(name);
  }
```

JAXWS

* Génération de clients Asynchrone
* https://docs.oracle.com/middleware/1213/wls/WSGET/jax-ws-async.htm#WSGET3408

  * Rappel ?
  * 2 modes: callback et polling

  * Callback : réponse traitée par le callback exécuté dans un thread. Réponse encapsulée dans Future
  * Polling: client sonde le serveur pour savoir si réponse est prête. Plus simple à coder mais plus lourd en ressource. Plus général ?

JAXWS

* Génération de clients Asynchrone
  * Exemple avec Callback

Param

```
cl.Serv_Service service = new cl.Serv_Service();
        cl.Serv port = service.getServPort();
        java.lang.String param = "test";
        javax.xml.ws.AsyncHandler<cl.HelloResponse>
asyncHandler = new
javax.xml.ws.AsyncHandler<cl.HelloResponse>() {
            public void
handleResponse(javax.xml.ws.Response<cl.HelloRespons
e> response) {
```

Réponse

```
try {
    System.out.println("Result = "+
response.get().getReturn());
            } catch(Exception ex) {
                // TODO handle exception
            }
        }
    };
    java.util.concurrent.Future<? extends
java.lang.Object> result = port.helloAsync(param,
asyncHandler);
    while(!result.isDone()) {
        // do something
        Thread.sleep(100);
    }
```

Appel

JAXWS

* Génération de clients Asynchrone
  * Exemple avec Polling
  * http://setgetweb.com/p/WAS9/ae/twbs_jaxwsclientasync.html

```
try {cl.Serv_Service service = new cl.Serv_Service();
        cl.Serv port = service.getServPort();
java.lang.String param = "test";
javax.xml.ws.Response<cl.HelloResponse> resp = port.helloAsync(param);
        while(!resp.isDone()) {
          // do something
          Thread.sleep(100);
        }
        System.out.println("Result = "+resp.get() getReturn());
      } catch (Exception ex) {
}
```

Stateful WS

JAXWS Statefull WS

2 possibilités :

- Par creation de session HTTP
  - Passage de la session dans les messages,
  - Plus légé au niveau du serveur

- Par l'annotation @stateful
  - Création d'instance au niveau du serveur

JAXWS Statefull WS

Création de session HTTP

https://docs.oracle.com/cd/E14571_01/web.1111/e13734/stateful.htm#WSADV234

Besoin de modifier le code du serveur ET le code du client

JAXWS Statefull WS

https://docs.oracle.com/cd/E14571_01/web.1111/e13734/stateful.htm#WSADV234
https://sujbiswa.wordpress.com/2011/03/15/ws-addressing-and-stateful-webservice/

Dans le code du service :

```
@WebService
public class ShoppingCart {
  @Resource    // Step 1
  private WebServiceContext wsContext;    // Step 2
  public int addToCart(Item item) {
    // Find the HttpSession
    MessageContext mc = wsContext.getMessageContext();    // Step 3
    HttpSession session =
((javax.servlet.http.HttpServletRequest)mc.get(MessageContext.SERVLET_REQUEST)).
getSession();
```

JAXWS Statefull WS

Suite du service (code exemple, depend de la logique à mettre en place)

```
if (session == null)
        throw new WebServiceException("No HTTP Session found");
// Get the cart object from the HttpSession (or create a new one)
List<Item> cart = (List<Item>)session.getAttribute("myCart");  // Step 4

if (cart == null)
        cart = new ArrayList<Item>();

// Add the item to the cart (note that Item is a class defined
// in the WSDL)
cart.add(item);
// Save the updated cart in the HTTPSession (since we use the same
// "myCart" name, the old cart object will be replaced)
session.setAttribute("myCart", cart);
// return the number of items in the stateful cart
return cart.size();
   }
}
```

JAXWS Statefull WS

Utilisation de la session chez le client :

```
ShoppingCart proxy = new CartService().getCartPort();
((BindingProvider)proxy).getRequestContext().put(BindingProvider.SESSION_MAINTAIN_PR
OPERTY, true);
// Create a new Item object with a part number of '123456' and an item
// count of 4.
Item item = new Item('123456', 4);
System.out.println(proxy.addToCart(item));
```

retourne 1

```
System.out.println(proxy.addToCart(item));
```

retourne 2

JAXWS Statefull WS

* Par annotation @stateful
* https://javaee.github.io/metro-jax-ws/doc/user-guide/release-documentation.html#users-guide-stateful-webservice
* Importer dans votre projet les lib jaxws et jaxb
* Ajouter le module adressing avec *@adressing*
  * Module qui je cite « provides transport-neutral mechanisms to **address** Web services and messages. »

JAXWS Statefull WS

* Par annotation @stateful
* Exemple:

```
@Stateful
@WebService(serviceName = "Serv")
@Addressing
public class Serv {
private String chaine;
  /**
   * This is a sample web service operation
   */
  @WebMethod(operationName = "addchaine")
  public synchronized String addchaine(@WebParam(name = "name") String txt) {// ou public static
    chaine = chaine + txt;
    return chaine ;
  }
}
```