

SOA – Services web étendus

Développer services web étendus



Mickaël BARON – 2010 (Rev. Janvier 2019)

Mickael BARON – 2010 (Rev. Janvier 2019)
mailto:baron.mickael@gmail.com ou mailto:baron@ensma.fr



@mickaelbaron



mickael-baron.fr

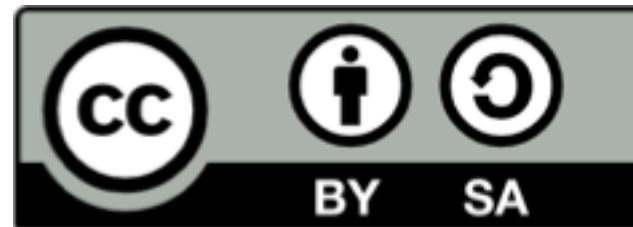
Licence

Creative Commons

Contrat Paternité

Partage des Conditions Initiales à l'Identique

2.0 France



<http://creativecommons.org/licenses/by-sa/2.0/fr>

À propos de l'auteur ...



➤ Mickaël BARON

➤ Ingénieur de Recherche au LIAS

- <https://www.lias-lab.fr> 
- Équipe : Ingénierie des Données et des Modèles
- Responsable des plateformes logicielles, « coach » technique



@mickaelbaron



mickael-baron.fr

➤ Responsable Rubriques Java de Developpez.com

- Communauté Francophone dédiée au développement informatique
- <https://java.developpez.com>
- 4 millions de visiteurs uniques et 12 millions de pages vues par mois
- 750 00 membres, 2 000 forums et jusqu'à 5 000 messages par jour



Plan du cours

- Généralités JAX-WS
- Développement serveur
 - Bottom -> Up
 - Top -> Down
- Développement client
- Annotations
- Handler
- Déploiement



Déroulement du cours

➤ Pédagogie du cours

- Illustration avec de nombreux exemples qui sont disponibles à l'adresse <http://mbaron.developpez.com/soa/jaxws/>
- Des bulles d'aide tout au long du cours
- Survol des principaux concepts en évitant une présentation exhaustive

➤ Logiciels utilisés

- Navigateur Web, Java et Maven



➤ Pré-requis

- Schema XML, WSDL, SOAP, JAXB

➤ Exemples

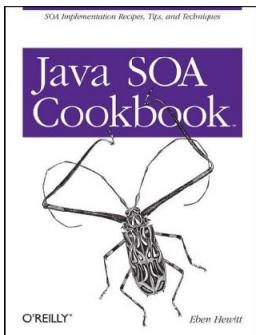
- <https://github.com/mickaelbaron/jaxws-examples>



Ressources

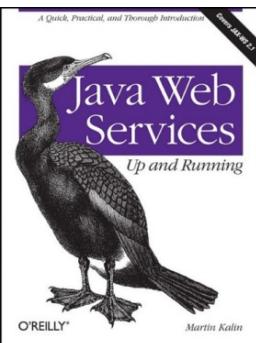
- Billets issus de Blog
 - blogs.sun.com/alexismpl/entry/metro_boulot_dodo
 - www.dotmyself.net/documentation/5.html
 - jee-bpel-soa.blogspot.com/search/label/jax-ws
- Articles
 - wiki.apache.org/ws/StackComparison
 - www.ibm.com/developerworks/webservices/library/ws-jsrart
 - java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
 - www.jroller.com/gmazza/entry/adding_jax_ws_handlers_to
 - blog.vinodsingh.com/2008/09/using-jax-ws-handlers_25.html
- Cours
 - java.sun.com/webservices/docs/1.6/tutorial/doc/index.html
 - www.javapassion.com/webservices/jaxwsbasics.pdf

Ressources : bibliothèque



➤ Java SOA Cookbook

- Auteur : Eben Hewitt
- Éditeur : Oreilly
- Edition : Mars 2009 - 752 pages - ISBN : 0596520727



➤ Java Web Services : Up and Running

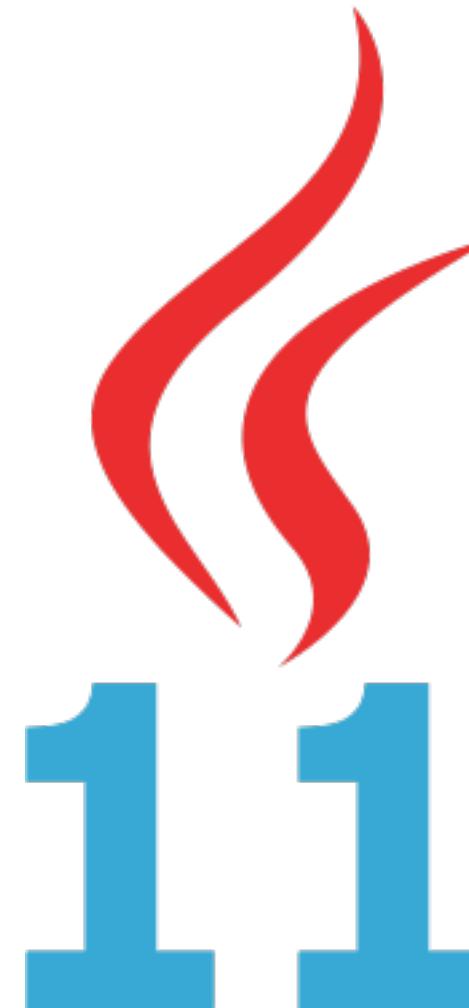
- Auteur : Martin Kalin
- Éditeur : Oreilly
- Edition : Février 2009 - 316 pages - ISBN : 059652112X



➤ Les Cahiers du Programmeur : Java EE 5

- Auteur : Antonio Goncalves
- Éditeur : Eyrolles
- Edition : Août 2008 - 351 pages - ISBN : 2212123639

Version Java supportée par ce support de cours



- Pas de gestion des modules dans les exemples (utilisation du classpath)

Généralités : développement de services web **SOAP**

- Ce cours s'intéresse au développement des services web de type **SOAP** ou **étendus**
 - Côté Serveur : code pour le traitement du service web
 - Côté Client : code qui permet d'appeler un service web
- La majorité des langages de programmation orientés web supportent le développement de services web
 - Java, PHP, C#, C++...
- Ce cours se limite au langage Java
- Différents *frameworks* de développement de services web
 - **JAX-WS** Specification (javaee.github.io/metro-jax-ws)
 - **AXIS 1 et 2** Apache (ws.apache.org/axis et ws.apache.org/axis2)
 - **CXF** Apache (cxf.apache.org)
 - **JBossWS** (jbossws.github.io/)

Généralités JAX-WS

- JAX-WS est l'acronyme Java API for XML Web Services
- JAX-WS est à la fois un standard et une implémentation
- La version courante est JAX-WS 2.3.2, précédemment
JAX-WS s'appelait JAX-RPC
- Code source API : github.com/eclipse-ee4j/jax-ws-api
- JAX-WS s'appuie sur un ensemble de JSR
 - JSR 224 : JAX-WS
 - JSR 109 : Web Services
 - JSR 181 : Web Services Metadata
 - JSR 222 : JAXB
 - JSR 250 : Common Annotations

Généralités JAX-WS : implémentation

- L'implémentation de référence est fournie par **METRO** appelée également JAX-WS RI (Reference Implementation)
 - Site projet Metro : *javaee.github.io/metro-jax-ws*
 - Code source : *github.com/eclipse-ee4j/metro-jax-ws*
 - Version actuelle : 2.3.2
- **WSIT** (Web Services Interoperability Technologies) est un complément pour gérer les services web avancés (WS-*
 - Site projet WSIT : *javaee.github.io/metro*
- L'implémentation JAX-WS est intégrée nativement à la JRE depuis la version 6 jusqu'à la version 10
- Possibilité de développer des services web sans serveur d'application (étudié à la fin)



Généralités JAX-WS : dépendances Maven

- À partir de Java 11, il faudra ajouter explicitement les dépendances

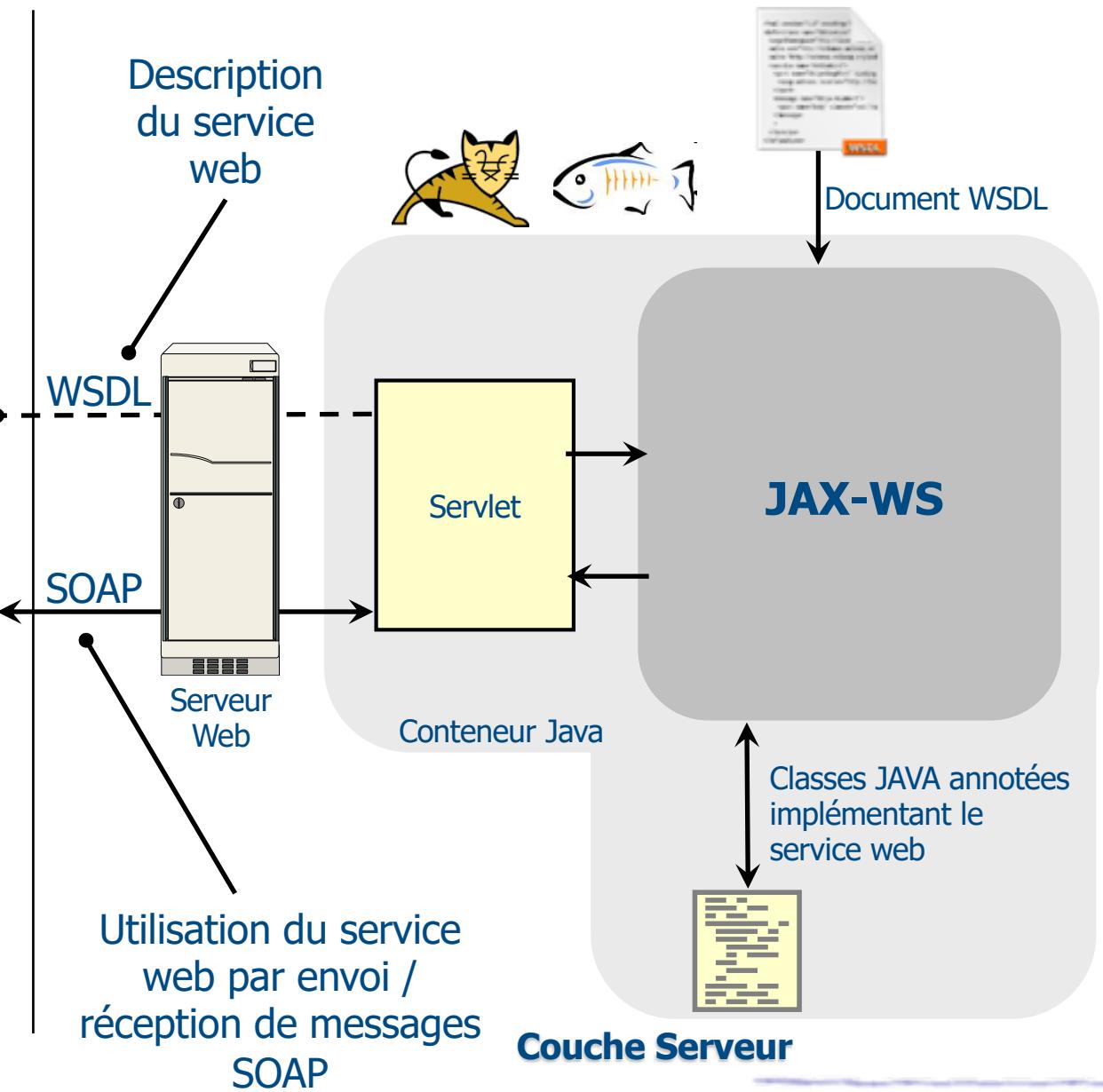
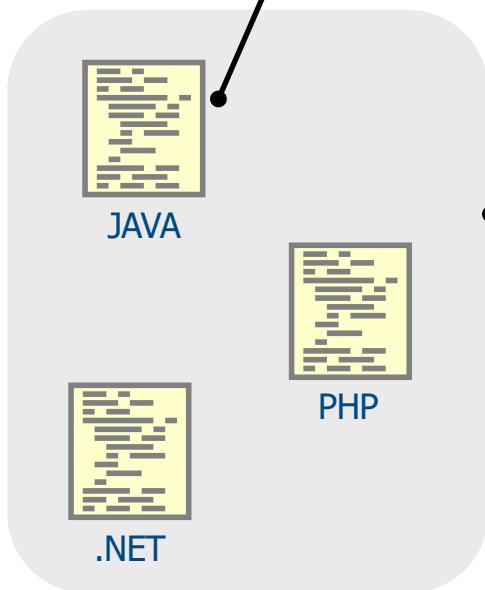
```
<dependency>
    <groupId>com.sun.xml.ws</groupId>
    <artifactId>jaxws-rt</artifactId>
    <version>2.3.2</version>
</dependency>
```

- Pour les versions antérieures à 2.3.2, il faudra ajouter une dépendance manquante

```
<dependency>
    <groupId>javax.activation</groupId>
    <artifactId>activation</artifactId>
    <version>1.1.1</version>
</dependency>
```

Généralités JAX-WS : fonctionnement

Développement de clients dans des langages différents



Généralités JAX-WS : fonctionnement

- Le développement de services web avec JAX-WS est basé sur des POJO (**P**lain **O**ld **J**ava **O**bject)
- Les fonctionnalités de base pour le développement de services web avec JAX-WS requiert simplement l'utilisation d'annotations Java
- Par conséquent aucun fichier de déploiement n'est requis
- Toutefois, les fonctionnalités avancées (appels asynchrones) nécessitent d'utiliser une API
- JAX-WS permet d'assurer l'indépendance du protocole (SOAP) et du transport (HTTP)

Plan du cours

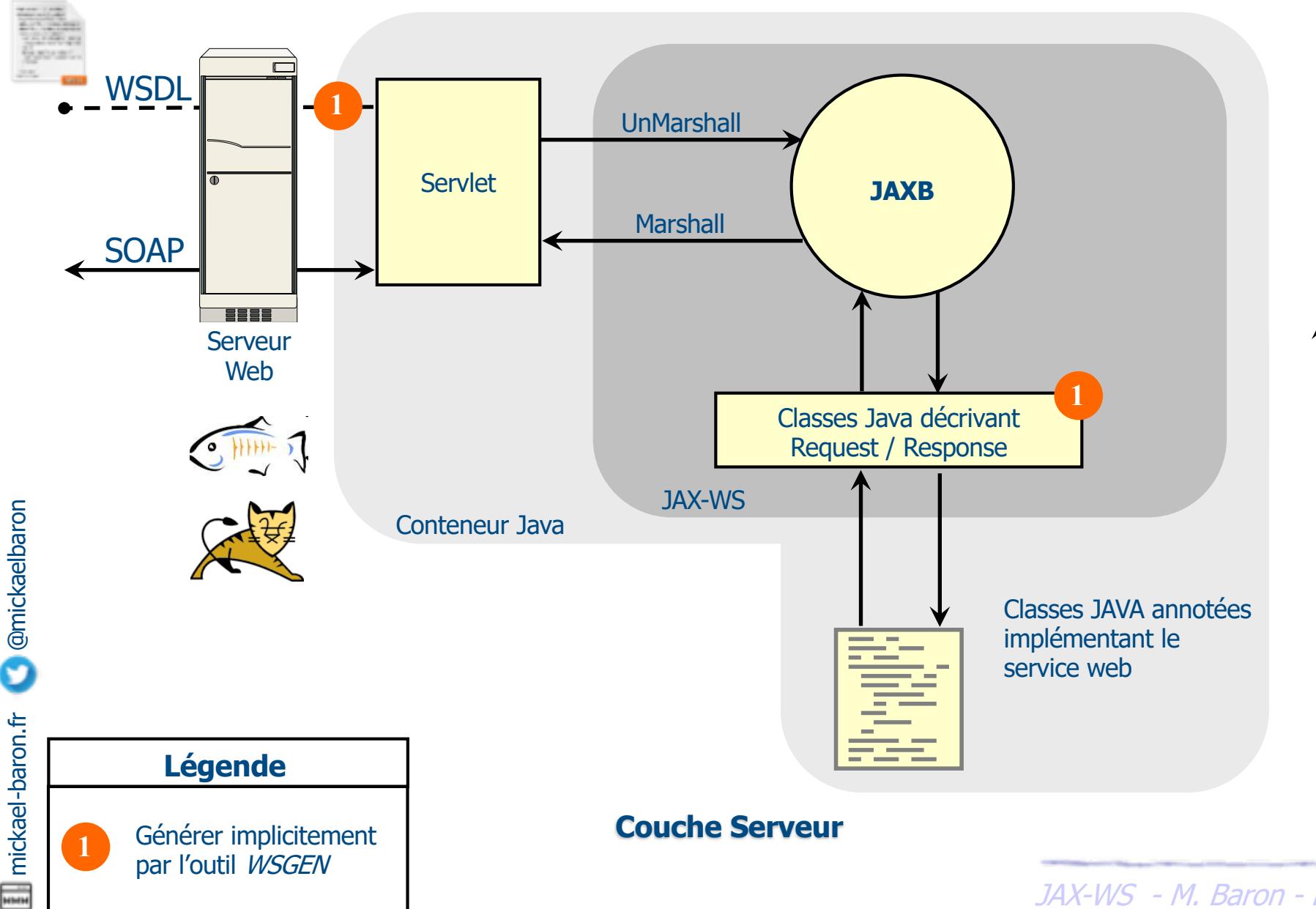
- Généralités JAX-WS
- Développement serveur
 - Bottom -> Up
 - Top -> Down
- Développement client
- Annotations
- Handler
- Déploiement



Développement Serveur : généralités

- Deux façons pour développer un service web avec JAX-WS
- **Approche Top / Down** (à partir d'un document WSDL)
 - Génération des différentes classes Java (JAXB et squelette du service web) en utilisant l'outil *wsimport*
 - Compléter le squelette de classe de l'implémentation
 - Compiler, déployer et tester
- **Approche Bottom / Up** (à partir d'un POJO)
 - Créer et annoter un POJO
 - Compiler, déployer et tester
 - Le document WSDL est automatiquement généré

Développement Serveur : Bottom / Up



Développement Serveur : Bottom / Up

- L'approche Bottom / Up consiste à démarrer le développement à partir d'une classe Java (POJO)
- Ajouter l'annotation *@WebService*
- Déployer l'application
- Le document WSDL est généré automatiquement en respectant les valeurs par défaut
 - URL du WSDL : *http://monserveur/app/Service?WSDL*
- Toutes les méthodes du POJO sont des opérations du service web
- La surcharge de méthodes n'est pas supportée

Développement Serveur : Bottom / Up

➤ Exemple : implémentation du service web *HelloWorld*

Le package doit être ajouté via
une dépendance

```
package fr.mickaelbaron.helloworldquietwebservice;  
  
import javax.jws.WebService;  
  
@WebService  
public class HelloWorldService {  
  
    public String makeHelloWorld(String value) {  
        return "Hello World to " + value;  
    }  
  
    public String simpleHelloWorld() {  
        return "Hello World to everybody";  
    }  
}
```

Deux opérations sont définies
dans la classe *HelloWorldService*

L'opération *makeHelloWorld*
contenant un message input et
un message output

L'opération *simpleHelloWorld*
contenant un message output
uniquement

HelloWorldService.java du projet
jaxws-helloworldquietwebservice

Développement Serveur : Bottom / Up

➤ Exemple (suite) : implémentation du service web *HelloWorld*



The screenshot shows a web browser window titled "Services Web". The address bar contains the URL "localhost:8080/helloworldquietwebservice/helloworldquiet". This URL is highlighted with a red box.

Adresse	Informations
Nom de service : (http://helloworldquietwebservice.mickaelbaron.fr/)HelloWorldServiceService Nom de port : (http://helloworldquietwebservice.mickaelbaron.fr/)HelloWorldServicePort	Adresse : http://localhost:8080/helloworldquietwebservice/helloworldquiet WSDL: http://localhost:8080/helloworldquietwebservice/helloworldquiet?wsdl Classe d'implémentation : fr.mickaelbaron.helloworldquietwebservice.HelloWorldService



The screenshot shows a web browser window titled "localhost:8080/helloworldquietwebservice/helloworldquiet?wsdl". This URL is highlighted with a red box.

```
</message>
<message name="simpleHelloWorldResponse">
<part name="parameters" element="tns:simpleHelloWorldResponse"/>
</message>
<message name="makeHelloWorld">
<part name="parameters" element="tns:makeHelloWorld"/>
</message>
<message name="makeHelloWorldResponse">
<part name="parameters" element="tns:makeHelloWorldResponse"/>
</message>
<portType name="HelloWorldService">
<operation name="simpleHelloWorld">
<input wsam:Action="http://helloworldquietwebservice.mickaelbaron.fr/HelloWorldService/simpleHelloWorldRequest"
message="tns:simpleHelloWorld"/>
<output wsam:Action="http://helloworldquietwebservice.mickaelbaron.fr/HelloWorldService/simpleHelloWorldResponse"
message="tns:simpleHelloWorldResponse"/>
</operation>
<operation name="makeHelloWorld">
<input wsam:Action="http://helloworldquietwebservice.mickaelbaron.fr/HelloWorldService/makeHelloWorldRequest"
message="tns:makeHelloWorld"/>
<output wsam:Action="http://helloworldquietwebservice.mickaelbaron.fr/HelloWorldService/makeHelloWorldResponse"
message="tns:makeHelloWorldResponse"/>
</operation>
</portType>
<binding name="HelloWorldServiceBinding" type="tns:HelloWorldServiceBindingType">
```

Document WSDL du service web développé

Développement Serveur : Bottom / Up

➤ Exemple : paramétrer le service web *HelloWorld*

```
package fr.mickaelbaron.helloworldwebservice;

@WebService(name="HelloWorld", targetNamespace="http://helloworldwebservice.mickaelbaron.fr/")
@SOAPBinding(style=Style.RPC, use=Use.LITERAL)
public interface HelloWorldService {

    @WebMethod(operationName="makeHelloWorld")
    @WebResult(name="helloWorldResult")
    String makeHelloWorld(
        @WebParam(name = "value")String value);

    @WebMethod(operationName="simpleHelloWorld")
    @WebResult(name="helloWorldResult")
    String simpleHelloWorld();
}
```

Utilisation d'une interface pour définir les paramètres du service web

HelloWorldService.java du projet **jaxws-helloworldwebservice**

```
package fr.mickaelbaron.helloworldwebservice;

@WebService(endpointInterface="fr.mickaelbaron.helloworldwebservice.HelloWorldService",
serviceName="HelloWorldService", portName="HelloWorldPort")
public class HelloWorldServiceImpl implements HelloWorldService {

    public String makeHelloWorld(String value) {
        return "Hello World to " + value;
    }

    public String simpleHelloWorld() {
        return "Hello World to everybody";
    }
}
```

Classe qui fournit l'implémentation du service web

HelloWorldServiceImpl.java du projet **jaxws-helloworldwebservice**

Développement Serveur : Bottom / Up

➤ Exemple (bis) : paramétrer le service web *HelloWorld*

```
package fr.mickaelbaron.helloworldwebservice;

@WebService(name="HelloWorld", targetNamespace="http://helloworldwebservice.mickaelbaron.fr/")
@SOAPBinding(style=Style.RPC, use=Use.LITERAL)
public interface HelloWorldService {

    @WebMethod(operationName="makeHelloWorld")
    @WebResult(name="helloWorldResult")
    String makeHelloWorld(
        @WebParam(name = "value")String value);

    @WebMethod(operationName="simpleHelloWorld")
    @WebResult(name="helloWorldResult")
    String simpleHelloWorld();
}
```

Utilisation d'une interface pour définir les paramètres du service web

HelloWorldService.java du projet **jaxws-helloworldwebservice**

```
package fr.mickaelbaron.helloworldwebservice;

@WebService(endpointInterface="fr.mickaelbaron.helloworldwebservice>HelloWorldService",
            serviceName="HelloWorldService", portName="HelloWorldPort")
public class HelloWorldServiceImpl {

    public String makeHelloWorld(String value) {
        return "Hello World to " + value;
    }

    public String simpleHelloWorld() {
        return "Hello World to everybody";
    }
}
```

Pas nécessaire d'indiquer l'implémentation puisqu'elle est précisée dans l'annotation (vérification à l'exécution)

Classe qui fournit l'implémentation du service web

HelloWorldServiceImpl.java du projet **jaxws-helloworldwebservice**

Développement Serveur : Bottom / Up

- Comme indiqué précédemment, du JDK 6 à 10 des API et des outils sont fournis pour manipuler des services web
- L'outil *wsgen* génère des *artifacts* (JAXB, WSDL) à partir de classes Java annotées via JAX-WS
- L'utilisation de cet outil n'est pas obligatoire puisque cette génération est implicite lors de l'exécution
- Exemples d'utilisation

```
wsgen -cp . fr.mickaelbaron.helloworldwebservice.HelloWorldServiceImpl -keep
```

Génère les classes Java annotées JAXB
(marshall et unmarshall)

```
wsgen -cp . fr.mickaelbaron.helloworldwebservice.HelloWorldServiceImpl -keep -wsdl
```

Génère également le document WSDL

Développement Serveur : Bottom / Up

➤ Exemple : *wsgen* avec Maven

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-maven-plugin</artifactId>
        <version>${jaxws.version}</version>
        <executions>
          <execution>
            <goals>
              <goal>wsgen</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <sei>fr.mickaelbaron.helloworldwebservice.HelloWorldServiceImpl</sei>
          <genWsdl>true</genWsdl>
          <keep>true</keep>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

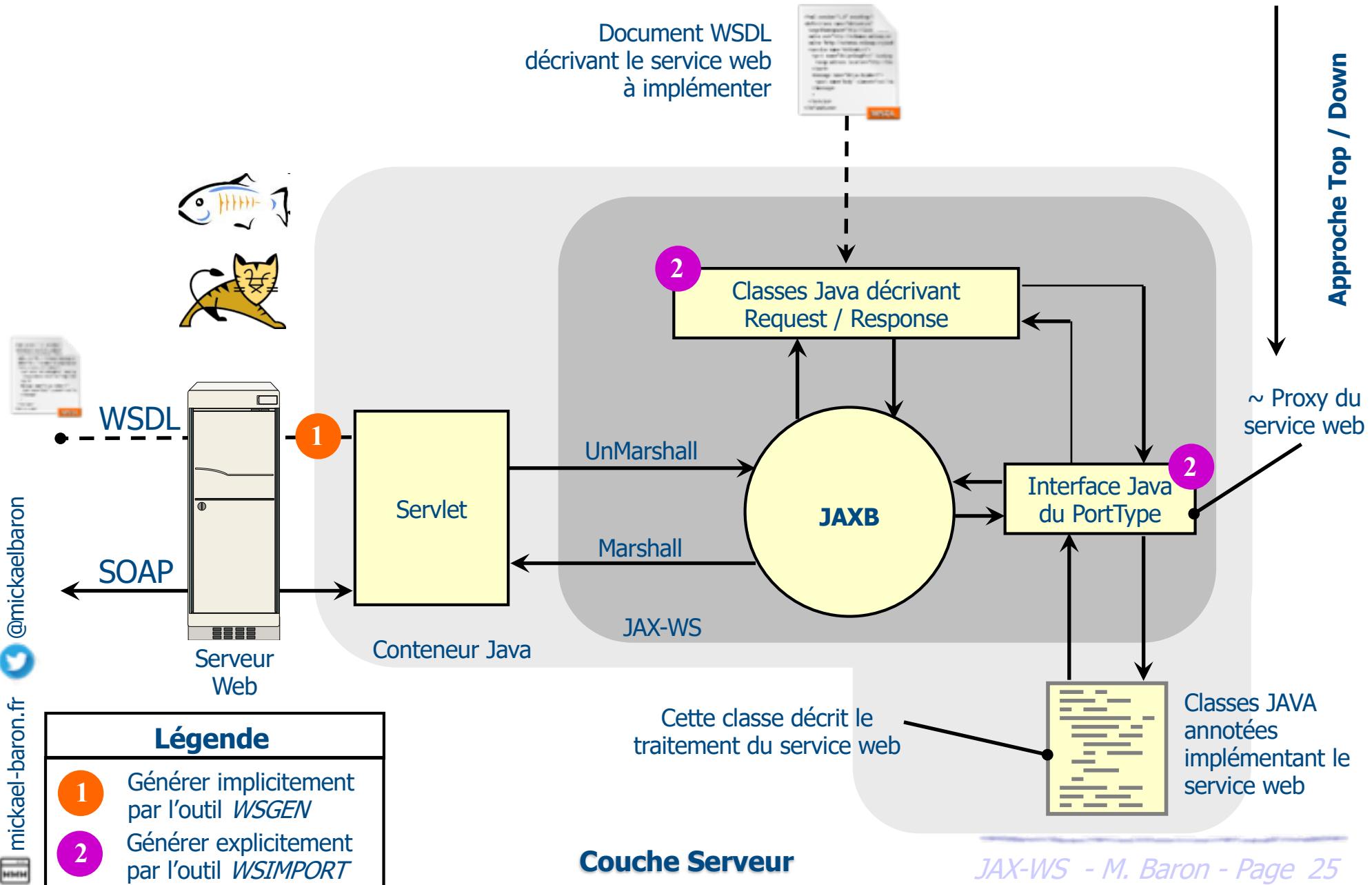
Plug-in à utiliser pour la génération des artifacts

Demande génération des ressources par rapport à la classe *HelloWorldService*

pom.xml du projet
jaxws-helloworldwebservice



Développement Serveur : Top / Down



Développement Serveur : Top / Down

- L'approche Top / Down consiste à démarrer le développement à partir d'un document WSDL
- Le document WSDL est accessible via une URL ou via un fichier physique
- Utilisation explicite de l'outil *wsimport* pour la génération du squelette du service web
 - Génération des classes liées à JAXB
 - Génération des interfaces WS (interface décrivant le *PortType*)
- Création d'un POJO annotée *@WebService* en précisant l'emplacement de l'interface du *portType* (Proxy)
- Déployer l'application sur un serveur d'application
- Le reste du processus de développement est identique à celui de l'approche Bottom / Up

Développement Serveur : Top / Down

- Exemple : développer le service web *Notebook* à partir du document WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions ...>
    <types>...</types>
    <portType name="NotebookService">
        <operation name="addPersonWithComplexType">
            <input message="tns:addPersonWithComplexType"/>
            <output message="tns:addPersonWithComplexTypeResponse"/>
        </operation>
        <operation name="getPersonByName">
            <input message="tns:getPersonByName"/>
            <output message="tns:getPersonByNameResponse"/>
        </operation>
        <operation name="getPersons">
            <input message="tns:getPersons"/>
            <output message="tns:getPersonsResponse"/>
        </operation>
        <operation name="addPersonWithSimpleType">
            <input message="tns:addPersonWithSimpleType"/>
        </operation>
    </portType>
    <binding name="NotebookPortBinding" type="tns:NotebookService">...</binding>
    <service name="NotebookService">...</service>
</definitions>
```



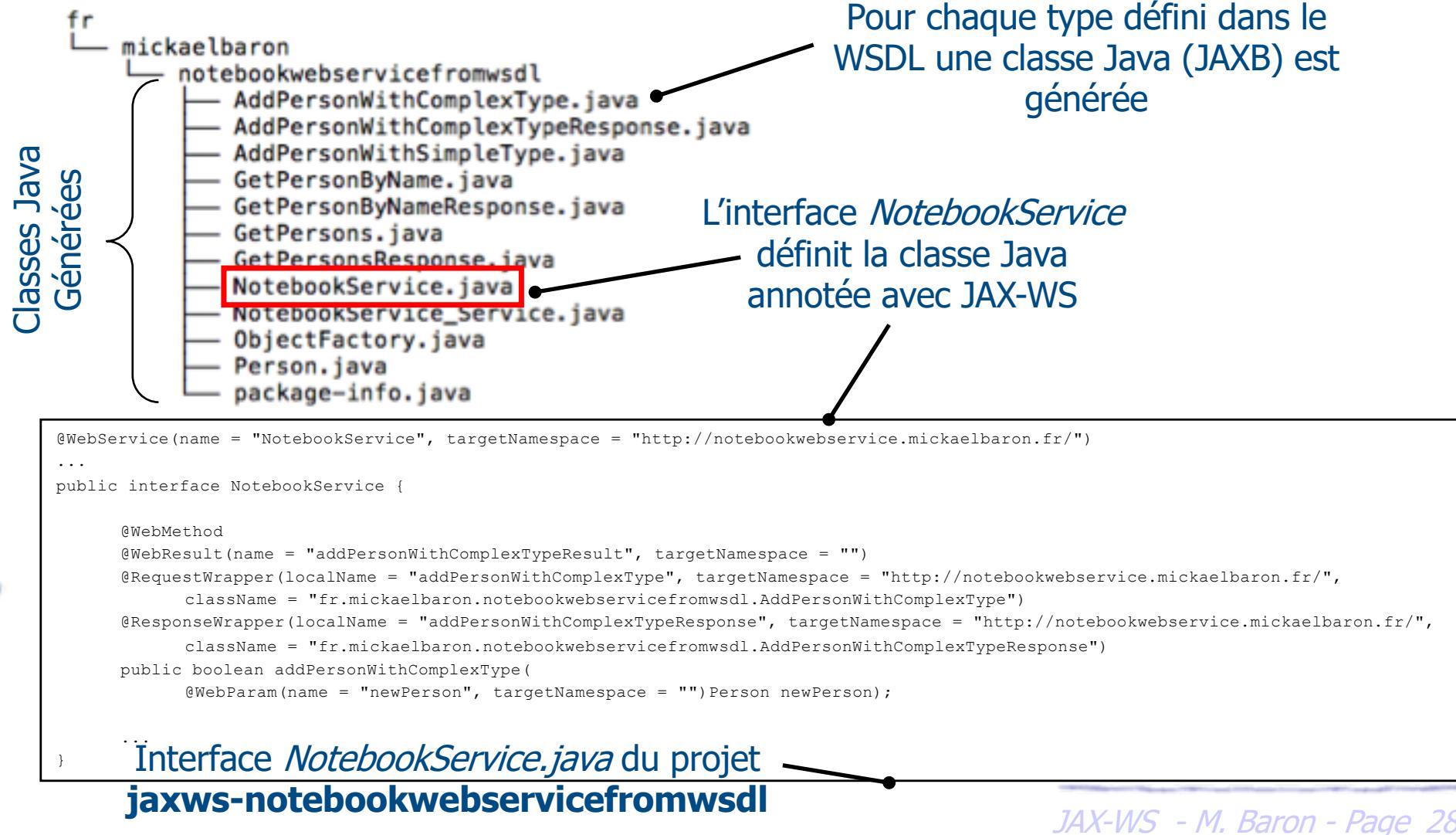
WSDL utilisé pour
générer le squelette
du service web

Document WSDL issu
du service web *Notebook*
(Projet **jaxws-notebookwebservice**)



Développement Serveur : Top / Down

- Exemple (suite) : développer le service web *Notebook* à partir du document WSDL



Développement Serveur : Top / Down

➤ Exemple (suite) : développer le service web *Notebook* à partir du document WSDL

```
@WebService(endpointInterface = "fr.mickaelbaron.notebookwebservicefromwsdl.NotebookService")  
public class NotebookServiceImpl {  
    public boolean addPersonWithComplexType(Person newPerson) {  
        ...  
        return true;  
    }  
  
    public Person getPersonByName(String name) {  
        Person current = new Person();  
        current.setName(name);  
        current.setBirthyear("1976");  
        current.setAddress("17 Square Mickael BARON");  
        return current;  
    }  
  
    public List<Person> getPersons() {  
        Person first = new Person();  
        first.setName("Ken BLOCK");  
        first.setBirthyear("1967");  
        first.setAddress("United States");  
        Person second = new Person();  
        second.setName("Colin MCRAE");  
        second.setBirthyear("1968");  
        second.setAddress("Scotland");  
  
        List<Person> tabPerson = new ArrayList<Person>();  
        tabPerson.add(first);  
        tabPerson.add(second);  
        return tabPerson;  
    }  
  
    public void addPersonWithSimpleType(String name, String address, String birthyear) {  
        System.out.println("Name : " + name + " Address : " + address + " birthyear : " + birthyear);  
    }  
}
```

L'utilisation du mot clé **implémentation** n'est pas obligatoire, l'annotation se charge d'effectuer cette relation

Implémentation du traitement du service web (Non généré à développer)

NotebookServiceImpl.java du projet
jaxws-notebookwebservicefromwsdl



Développement Serveur : Top / Down

➤ Exemple : *wsimport* avec Maven (wsdl à partir d'un fichier)

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-maven-plugin</artifactId>
        <version>${jaxws.version}</version>
        <executions>
          <execution>
            <goals><goal>wsimport</goal></goals>
          </execution>
        </executions>
        <configuration>
          <wsdlDirectory>${project.basedir}/src/wsdl</wsdlDirectory>
          <sourceDestDir>${project.build.directory}/generated-sources/jaxws-wsimport</sourceDestDir>
          <keep>true</keep>
          <packageName>fr.mickaelbaron.notebookwebservicefromwsdl</packageName>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Plug-in à utiliser pour la génération des classes Java

Le document WSDL disponible depuis **un fichier**

Package par défaut des classes générées

pom.xml du projet

jaxws-notebookwebservicefromwsdl

Développement Serveur : Top / Down

➤ Exemple : *wsimport* avec Maven (wsdl à partir d'une URL)

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-maven-plugin</artifactId>
        <version>${jaxws.version}</version>
        <executions>
          <execution>
            <goals><goal>wsimport</goal></goals>
          </execution>
        </executions>
        <configuration>
          <wsdlUrls>
            <wsdlUrl>http://localhost:8080/notebookwebservice/notebook?wsdl</wsdlUrl>
          </wsdlUrls>
          <sourceDestDir>${project.build.directory}/generated-sources/jaxws-wsimport</sourceDestDir>
          <keep>true</keep>
          <packageName>fr.mickaelbaron.notebookwebservicefromwsdl</packageName>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Plug-in à utiliser pour la génération des classes Java

Le document WSDL disponible depuis **une URL**

Package par défaut des classes générées

pom.xml du projet

jaxws-notebookwebservicefromwsdl

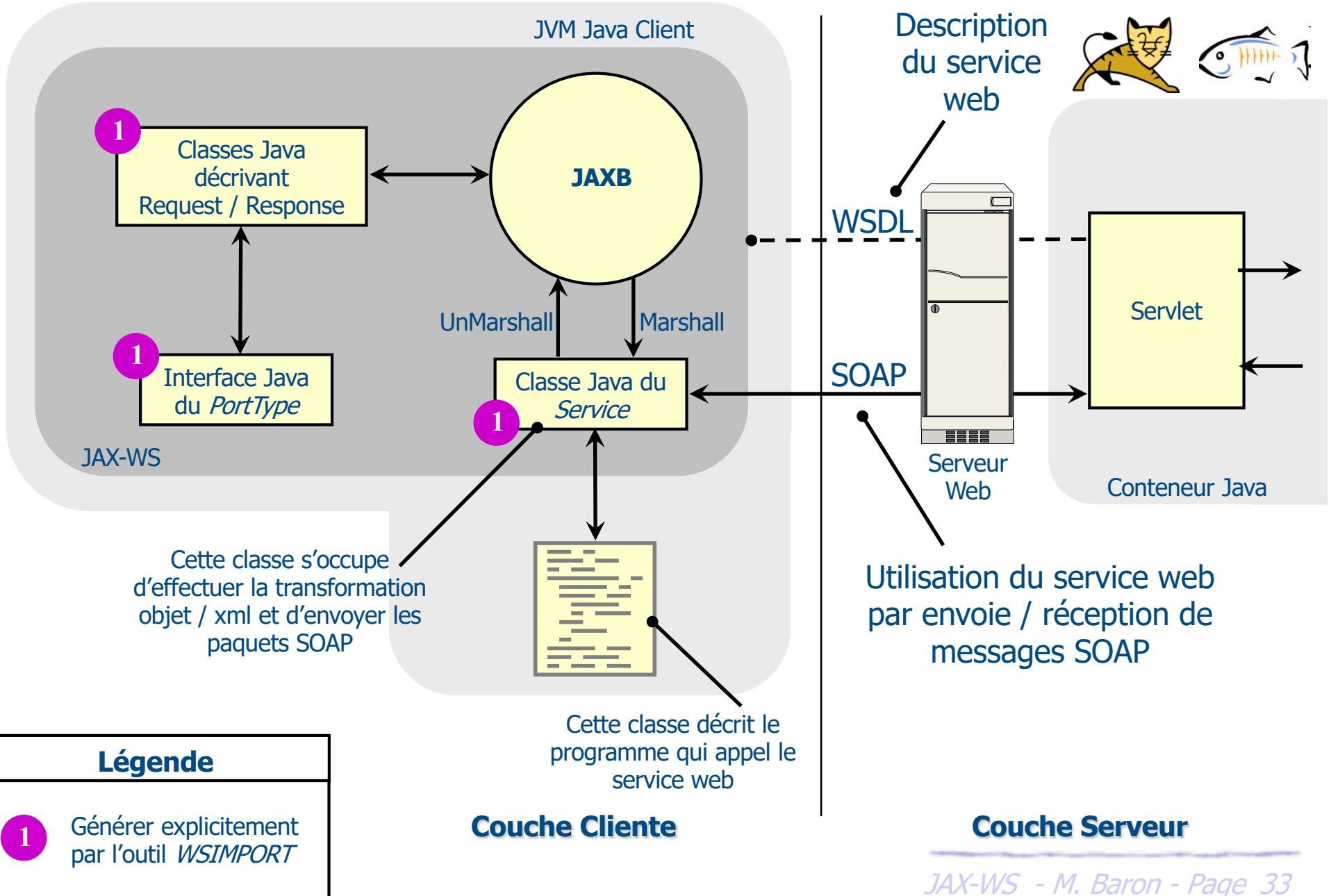


Plan du cours

- Généralités JAX-WS
- Développement serveur
 - Bottom -> Up
 - Top -> Down
- **Développement client**
- Annotations
- Handler
- Déploiement



Développement Client Java



Développement Client Java

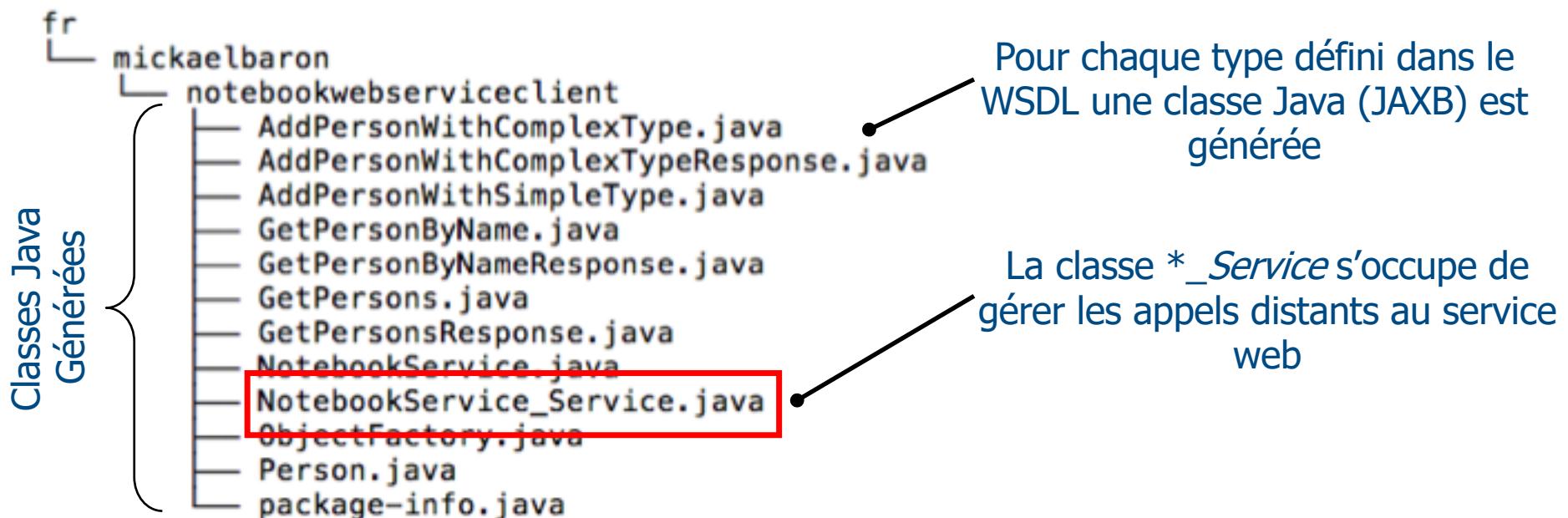
- Le développement du client consiste à appeler des opérations du service web à partir d'un programme Java
- Le client peut être une application développée
 - Java SE (JavaFX, Swing, Eclipse RCP)
 - Jakarta EE avec les EJB (voir section EJB)
- Possibilité de générer des appels aux services web de manière synchrone et asynchrone
- Le développeur ne manipule que du code Java, le code XML est caché (JAXB)

Développement Client Java

- Le développement du client suit une procédure similaire à l'approche Top / Down où le point de départ est le document WSDL (via une URL ou via un fichier physique)
- Utilisation explicite de l'outil *wsimport* pour la génération du squelette du service web
 - Génération des classes liées à JAXB
 - Génération de classes service web (*PortType* et *Service*)
- Création d'une instance de la classe *Service*
- Récupération d'un port via *get<ServiceName>Port()*
- Invocation des opérations

Développement Client Java

- Exemple : développer un client Java (Synchrone) pour le service web *Notebook*



Le résultat de la génération est identique à la génération effectuée pour l'approche Top / Down



Développement Client Java

- Exemple (suite) : développer un client Java (Synchrone) pour le service web *Notebook*

```
public class NotebookServiceClient {  
    public static void main(String[] args) {  
        NotebookService_Service notebookService = new NotebookService_Service();  
        NotebookService notebookPort = notebookService.getNotebookPort();  
  
        Person refPerson = new Person();  
        refPerson.setName("Baron Mickael");  
        refPerson.setAddress("Poitiers");  
        refPerson.setBirthyear("1976");  
  
        boolean addPersonWithComplexType = notebookPort.addPersonWithComplexType(refPerson);  
  
        System.out.println(addPersonWithComplexType);  
    }  
}
```

Création d'une instance de la classe *Service*

Récupération d'un port via *getNotebookPort()*

NotebookServiceClient.java du projet
jaxws-notebookwebserviceclient

Appel de l'opération du service web



Déployer le service web du projet
jaxws-notebookwebservice avant
l'exécution de ce projet

Développement Client Java

- Exemple (suite) : développer un client Java (Synchrone) pour le service web *Notebook*

```
@WebService(endpointInterface = "fr.mickaelbaron.notebookwebservice.NotebookService")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        System.out.println("Starting process ...");

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Name: " + newPerson.getName() + " Address: " + newPerson.getAddress() +
                           " birthyear: " + newPerson.getBirthyear());

        return true;
    }
    ...
}
```

Le traitement du service web peut prendre un certain temps

NotebookServiceImpl.java du projet **jaxws-notebookwebservice**

Dans le cas où le temps de réponse d'un service web est particulièrement long, prévoir un appel asynchrone



Développement Client Java

- Si le temps de réponse de l'appel à une opération d'un service web est long, prévoir un appel asynchrone
- JAX-WS permet d'appeler des services web en mode asynchrone si l'information est précisée dans le binding
- Lors de la génération des fichiers classes (via *wsimport*) fournir un fichier binding suivant

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bindings wsdlLocation="http://localhost:8080/NotebookWebService/notebook?wsdl"
    xmlns="http://java.sun.com/xml/ns/jaxws">
    <enableAsyncMapping>true</enableAsyncMapping>
</bindings>
```

binding.xml du projet

jaxws-notebookwebserviceasyncclient

Développement Client Java

- Exemple : développer un client Java (Asynchrone) pour le service web *Notebook*

```
<project ...>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>com.sun.xml.ws</groupId>
        <artifactId>jaxws-maven-plugin</artifactId>
        <version>${jaxws.version}</version>
        <executions>
          <execution>
            <goals><goal>wsimport</goal></goals>
          </execution>
        </executions>
        <configuration>
          <wsdlUrls>
            <wsdlUrl>http://localhost:8080/notebookwebservice/notebook?wsdl</wsdlUrl>
          </wsdlUrls>
          <sourceDestDir>${project.build.directory}/generated-sources/jaxws-wsimport</sourceDestDir>
          <keep>true</keep>
          <packageName>fr.mickaelbaron.notebookwebserviceasyncclient</packageName>
          <bindingFiles>
            <bindingFile>
              ${basedir}/src/jaxws/binding.xml
            </bindingFile>
          </bindingFiles>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

Plug-in à utiliser pour la génération des classes Java

Le document WSDL disponible depuis **une URL**

Permet de spécifier le fichier *binding.xml*

pom.xml du projet

jaxws-notebookwebserviceasyncclient

Développement Client Java

➤ Exemple (suite) : développer un client Java (Asynchrone) pour le service web *Notebook*

```
public class NotebookServiceAsyncClient {  
    public static void main(String[] args) throws InterruptedException {  
        Notebook_Service notebook_Service = new Notebook_Service();  
        Notebook noteBookPort = notebook_Service.getNoteBookPort();  
  
        Person refPerson = new Person();  
        refPerson.setName("Baron Mickael");  
        refPerson.setAddress("Poitiers");  
        refPerson.setBirthyear("1976");  
  
        noteBookPort.addPersonWithComplexTypeAsync(refPerson, new AsyncHandler<AddPersonWithComplexTypeResponse>() {  
            public void handleResponse(Response<AddPersonWithComplexTypeResponse> res) {  
                if (!res.isCancelled() && res.isDone()) {  
                    try {  
                        AddPersonWithComplexTypeResponse value = res.get();  
                        System.out.println(value.isAddPersonWithComplexTypeResult());  
                    } catch (Exception e) {  
                        e.printStackTrace();  
                    }  
                }  
            }  
        });  
        Thread.sleep(10000);  
        System.out.println("Terminé");  
    }  
}
```

Des méthodes spécifiques au mode asynchrone sont générées

Déclenché quand le traitement de l'opération sera terminé

Pour éviter de terminer le programme

Déployer le service web du projet **jaxws-notebookwebservice** avant l'exécution de ce projet

Plan du cours

- Généralités JAX-WS
- Développement serveur
 - Bottom -> Up
 - Top -> Down
- Développement client
- **Annotations**
- Handler
- Déploiement



Annotations : généralités

- JAX-WS repose sur l'utilisation massive d'annotations pour la configuration d'un service web
- JSR utilisées (JSR 224, 222, 181 et 250)
- Les principales annotations sont les suivantes
 - *@WebService* : POJO implémentant un service web
 - *@WebMethod* : Paramétrer une opération
 - *@WebParam* : Paramétrer un message
 - *@WebResult* : Paramétrer un message de sortie
 - *@WebFault* : Paramétrer un message fault
- À noter que seule l'utilisation de l'annotation *@WebService* est nécessaire (utilisation de valeurs par défaut)
- Nous détaillerons chacune des annotations de manière à découvrir les paramètres disponibles

Annotations : @WebService

- Annote une classe Java pour définir l'implémentation du service web
- Annote une interface Java pour définir la description du service web
- Attributs de l'annotation *@WebService*
 - *String name* : nom du service web
 - *String endpointInterface* : nom de l'interface décrivant le service web
 - *String portName* : nom du port
 - *String serviceName* : nom du service du service web
 - *String targetNamespace* : le namespace du service web
 - *String wsdlLocation* : l'emplacement du WSDL décrivant le Service Web

Annotations : @WebMethod

- Annote une méthode d'une classe Java exposée comme une opération du service web
- Attributs de l'annotation : *@WebMethod*
 - *String action* : l'action de l'opération. Dans le cas d'un binding SOAP, cela détermine la valeur de l'action SOAP
 - *boolean exclude* : précise que la méthode ne doit pas être exposée comme une opération. Ne pas utiliser dans une interface Java
 - *String operationName* : précise le nom de l'attribut *name* défini dans l'élément *operation* du document WSDL

Annotations : @WebParam

- Décrit la relation entre un paramètre d'entrée d'une méthode et un message part d'une opération
- Attributs de l'annotation
 - *boolean header* : précise si le paramètre doit être transmis dans l'en-tête du message (*true*) ou dans le corps (*false*)
 - *WebParam.Mode mode* : précise le type d'accès au paramètre (*IN*, *OUT* ou *INOUT*)
 - *String name* : nom du paramètre
 - *String partName* : le nom du *wsdl:part* représentant ce paramètre
 - *String targetNamespace* : l'espace de nommage de ce paramètre

Annotations : @WebResult

- Décrit la relation entre le paramètre de sortie d'une méthode et un message part d'une opération
- Attributs de l'annotation
 - *boolean header* : précise si le paramètre de sortie doit être transmis dans l'en-tête du message (*true*) ou dans le corps (*false*)
 - *String name* : nom du paramètre de sortie
 - *String partName* : le nom du *wsdl:part* représentant ce paramètre de sortie
 - *String targetNamespace* : l'espace de nommage de ce paramètre de sortie

Plan du cours

- Généralités JAX-WS
- Développement serveur
 - Bottom -> Up
 - Top -> Down
- Développement client
- Annotations
- **Handler**
- Déploiement



Handler : généralités

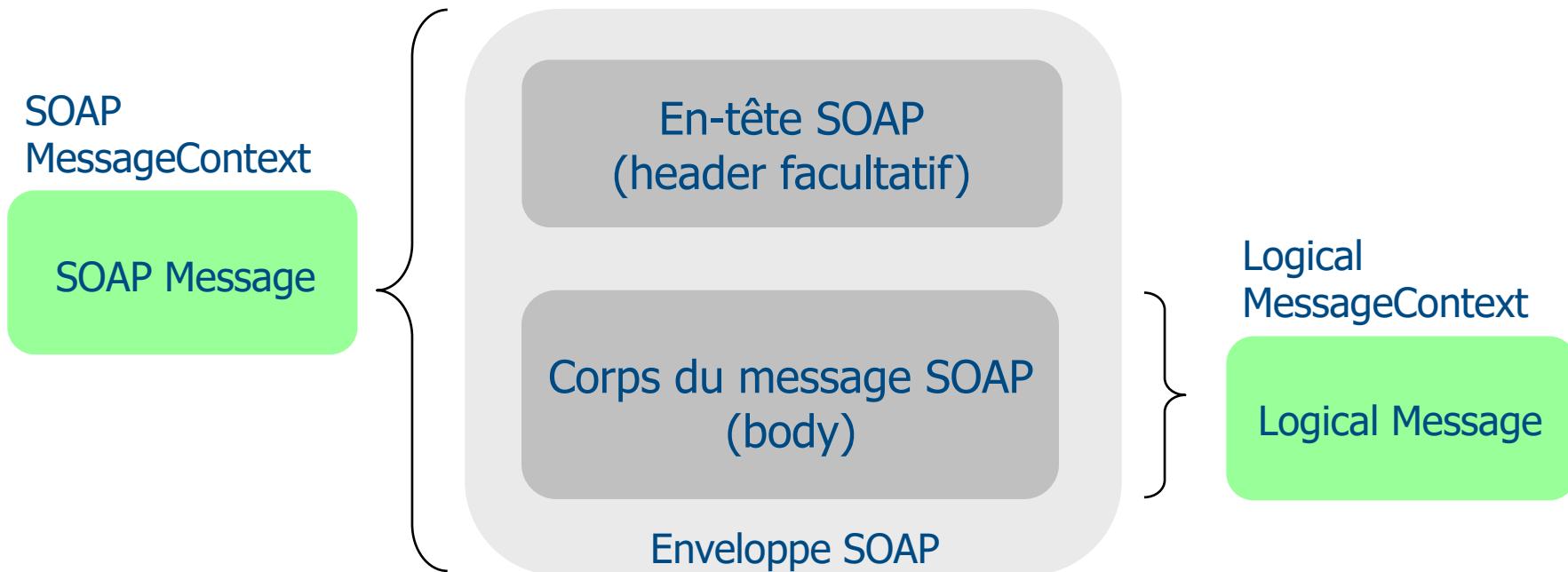
- Les « handlers » sont des intercepteurs permettant de réaliser des traitements lors de la réception et l'émission de messages
 - Lors de la réception ils sont déclenchés avant l'appel à une opération
 - Lors de l'émission ils sont déclenchés après l'appel à une opération
- Un « handler » est disponible dans la couche JAX-WS et par conséquent autant sur la partie cliente que celle du serveur
- Quand l'utiliser ?
 - Pour filtrer les appels aux opérations d'un service web
 - Pour l'écriture des logs
- Deux types de « handlers »
 - Handlers liés au protocole de transport (ex : SOAP)
 - Handlers liés au contenu transféré appelé *logical handlers* qui est indépendant du protocole

Handler : généralités

- JAX-WS définit l'interface *Handler* contenant les principales méthodes
 - *boolean handleMessage(C context)* : invoquée lors des messages entrants et sortants. Si false est retourné le processus est arrêté
 - *boolean handleFault(C context)* : invoquée lors des messages en erreur (fault)
- Le type générique *Chérite* de *MessageContext* qui est une *Map* contenant un ensemble de propriétés
 - *MESSAGE_OUTBOUND_PROPERTY* : pour savoir s'il s'agit de messages entrants ou sortants
 - *HTTP_REQUEST_HEADERS* : pour récupérer l'en-tête HTTP de la requête
 - *WSDL_OPERATION* : nom de l'opération WSDL
 - *WSDL_SERVICE* : nom du service WSDL

Handler : généralités

- Deux sous interfaces à *handler* sont proposées pour décrire chaque type de « handler »
- ***SOAPHandler*** a un accès sur l'intégralité du message SOAP incluant les en-têtes
- ***LogicalHandler*** pour une indépendance du protocole de transport et un accès sur le contenu du message



Handler : côté serveur

- Développement d'une classe Java qui implémente soit l'interface *SOAPHandler* soit *LogicalHandler*
- Définir un fichier de correspondance (*handlers.xml*) qui précise les classes implémentant les handlers

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-name>NOM DU HANDLER</handler-name>
            <handler-class>CLASSE DU HANDLER</handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```



- Ajouter l'annotation *@HandlerChain* sur le POJO du service web

Handler : côté serveur

- Exemple : ajouter un *handler* de type *SOAPHandler* sur le service web *Notebook*

```
public class SOAPLoggingHandler implements SOAPHandler<SOAPMessageContext> {

    private static PrintStream out = System.out;

    public Set<QName> getHeaders() { return null; }

    public void close(MessageContext context) { }

    public boolean handleMessage(SOAPMessageContext smc) {
        logToSystemOut(smc);
        return true;
    }

    public boolean handleFault(SOAPMessageContext smc) {
        logToSystemOut(smc);
        return true;
    }

    private void logToSystemOut(SOAPMessageContext smc) {
        Boolean outboundProperty = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        if (outboundProperty.booleanValue()) {
            out.println("\nOutgoing message from web service provider:");
        } else {
            out.println("\nIncoming message to web service provider:");
        }
        SOAPMessage message = smc.getMessage();
        try {
            message.writeTo(out);
            out.println("");
        } catch (Exception e) {
            out.println("Exception in handler: " + e);
        }
    }
}
```

Handler : côté serveur

- Exemple (suite) : ajouter un *handler* de type *SOAPHandler* sur le service web *Notebook*

handlers.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
    <handler-chain>
        <handler>
            <handler-name>fr.mickaelbaron.notebookwebservicewithsoaphandler.SOAPLoggingHandler</handler-name>
            <handler-class>fr.mickaelbaron.notebookwebservicewithsoaphandler.SOAPLoggingHandler</handler-class>
        </handler>
    </handler-chain>
</handler-chains>
```

```
@WebService(endpointInterface = "fr.mickaelbaron.notebookwebservicefromwsdl.Notebook")
@HandlerChain(file = "handlers.xml")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        ...
    }

    public Person getPersonByName(String name) {
        ...
    }

    public List<Person> getPersons() {
        ...
    }

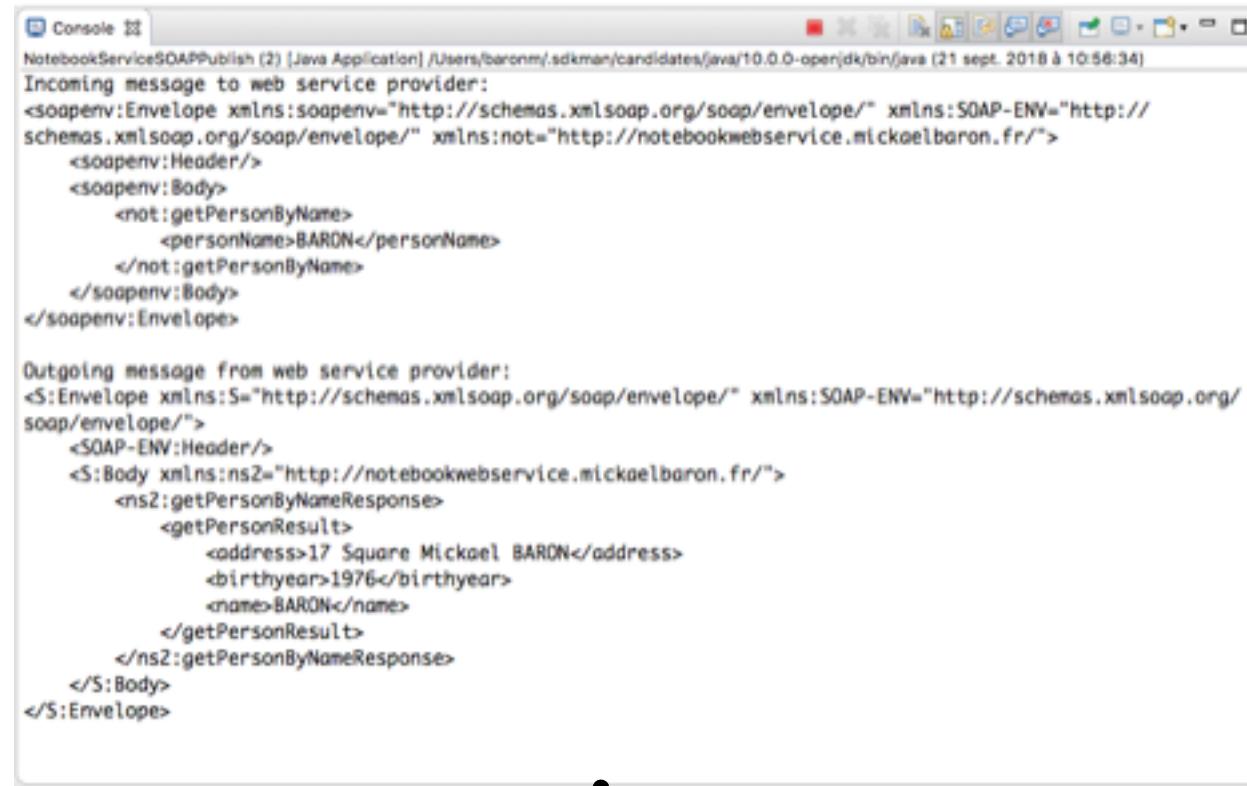
    public void addPersonWithSimpleType(String name, String address, String birthyear) {
        ...
    }
}
```

NotebookServiceImpl.java du projet
jaxws-notebookwebservicewithsoaphandler



Handler : côté serveur

- Exemple (suite) : ajouter un *handler* de type *SOAPHandler* sur le service web *Notebook*



The screenshot shows a Java application window titled "Console". The output pane displays two SOAP messages. The first message is an incoming request from the client to the web service provider:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://notebookwebservice.mickaelbaron.fr/">
    <soapenv:Header/>
    <soapenv:Body>
        <ns:getPersonByName>
            <personName>BARON</personName>
        </ns:getPersonByName>
    </soapenv:Body>
</soapenv:Envelope>
```

The second message is an outgoing response from the web service provider to the client:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <S:Body xmlns:ns2="http://notebookwebservice.mickaelbaron.fr/">
        <ns2:getPersonByNameResponse>
            <getPersonResult>
                <address>17 Square Mickael BARON</address>
                <birthyear>1976</birthyear>
                <name>BARON</name>
            </getPersonResult>
        </ns2:getPersonByNameResponse>
    </S:Body>
</S:Envelope>
```

Les messages entrants et sortants sont affichés avec la structure de l'enveloppe SOAP

Handler : côté client

- Exemple : ajouter un *handler* de type *SOAPHandler* sur le client du service web *Notebook*

```
public class SOAPLoggingHandler implements SOAPHandler<SOAPMessageContext> {

    private static PrintStream out = System.out;

    public Set<QName> getHeaders() { return null; }

    public void close(MessageContext context) { }

    public boolean handleMessage(SOAPMessageContext smc) {
        logToSystemOut(smc);
        return true;
    }

    public boolean handleFault(SOAPMessageContext smc) {
        logToSystemOut(smc);
        return true;
    }

    private void logToSystemOut(SOAPMessageContext smc) {
        Boolean outboundProperty = (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        if (outboundProperty.booleanValue()) {
            out.println("\nIncoming message from web service client:");
        } else {
            out.println("\nOutgoing message to web service client:");
        }
        SOAPMessage message = smc.getMessage();
        try {
            message.writeTo(out);
            out.println("");
        } catch (Exception e) {
            out.println("Exception in handler: " + e);
        }
    }
}
```

SOAPLoggingHandler.java du projet
jaxws-notebookwebserviceclientwithsoaphandler

Handler : côté client

- Exemple (suite) : ajouter un *handler* de type *SOAPHandler* sur le client du service web *Notebook*

```
public class NotebookServiceAsyncClient {  
    public static void main(String[] args) {  
        NotebookService_Service notebookService = new NotebookService_Service();  
        NotebookService noteBookPort = notebookService.getNotebookPort();  
  
        List<Handler> myHandler = new ArrayList<Handler>();  
        myHandler.add(new SOAPLoggingHandler());  
  
        ((BindingProvider)noteBookPort).getBinding().setHandlerChain(myHandler);  
  
        Person refPerson = new Person();  
        refPerson.setName("Baron Mickael");  
        refPerson.setAddress("Poitiers");  
        refPerson.setBirthyear("1976");  
        boolean addPersonWithComplexType = noteBookPort.addPersonWithComplexType(refPerson);  
        System.out.println(addPersonWithComplexType);  
    }  
}
```

NotebookClient.java du projet
jaxws-notebookwebserviceclientwithsoaphandler



Plan du cours

- Généralités JAX-WS
- Développement serveur
 - Bottom -> Up
 - Top -> Down
- Développement client
- Annotations
- Handler
- Déploiement



Déploiement

- Trois formes de déploiement pour exécuter votre service web
 - **Déploiement pour les tests** (mode bac à sable)
 - **Déploiement sur un serveur d'application Java**
 - Avant l'arrivée des microservices
 - Nécessite l'installation d'un serveur (Jetty, Glassfish...)
 - **Déploiement comme une application Java classique**
 - Populaire depuis l'arrivée des microservices
 - Serveur d'application est intégré (embedded)

Déploiement pour les tests

- Le développement serveur est possible à partir d'une application Java classique puisque l'API JAX-WS intègre un serveur web embarqué
 - Pas de dépendance supplémentaire (Tomcat, Jetty, Glassfish)
 - Le déploiement est immédiat
- Les deux approches (*Top / Down* et *Bottom / Up*) sont applicables à ce mode de développement
- Usages
 - Pour les tests fonctionnels, fournir des Mock de services web
 - Fournir des services web à une application type client lourd

Déploiement pour les tests

- Développer une classe Java implémentant le service web
(ou à partir d'un WSDL et en utilisant **wsimport**)
- Ajouter l'annotation *@WebService*
- Créer explicitement une instance du POJO
- Publier le service web par l'intermédiaire de la méthode

```
Endpoint Endpoint.publish(String adresse, Object implementor)
```

- Le document WSDL est généré automatiquement à l'exécution de l'application
- URL du WSDL : *http://monserveur/service?WSDL*
- Toutes les méthodes du POJO sont des opérations du service web

Déploiement pour les tests

- La méthode *publish* est utilisée pour démarrer la publication du service web
 - *String adresse* : adresse de déploiement du service web depuis le serveur Web embarqué (*http://localhost:8080/ws*)
 - *Object implementor* : instance de l'implémentation du service web
- Différentes méthodes proposées par la classe *Endpoint* (type de retour de la méthode *publish*)
 - *boolean isPublished()* : vérifie si le service est en publication
 - *void stop()* : arrête la publication du service web

Déploiement pour les tests

➤ Exemple : publier le service web Notebook

```
@WebService(name="NotebookService",targetNamespace="http://notebookwebservice.mickaelbaron.fr/")
public interface NotebookService {
    @WebMethod(operationName="addPersonWithComplexType")
    @WebResult(name="addPersonWithComplexTypeResult")
    boolean addPersonWithComplexType(
        @WebParam(name = "newPerson") Person newPerson);
    ...
}
```

Interface *NotebookService.java* du projet
jaxws-notebookwebservice

```
@WebService(endpointInterface="fr.mickaelbaron.notebookwebservice.NotebookService",
            serviceName="NotebookService",
            portName="NotebookPort")
public class NotebookServiceImpl implements NotebookService {
    @Override
    public boolean addPersonWithComplexType(Person newPerson) { ... }
    @Override
    public Person getPersonByName(String name) { ... }
    @Override
    public Person[] getPersons() { ... }
    @Override
    public void addPersonWithSimpleType(String name, String address, String birthyear) { ... }
}
```

Classe *NotebookServiceImpl.java* du projet
jaxws-notebookwebservice

Déploiement pour les tests

➤ Exemple (suite) : publier le service web

```
public class NotebookServicePublish extends JFrame {  
    private Endpoint publish;  
    public NotebookServicePublish() {  
        final JButton startPublish = new JButton();  
        startPublish.setText("Start Publish");  
        final JButton stopPublish = new JButton();  
        stopPublish.setText("Stop Publish");  
  
        startPublish.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                NotebookService current = new NotebookServiceImpl();  
                publish = Endpoint.publish("http://localhost:8080/notebookwebservice/notebook", current);  
                startPublish.setEnabled(false);  
                stopPublish.setEnabled(true);  
            }  
        });  
  
        stopPublish.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                publish.stop();  
                stopPublish.setEnabled(false);  
                startPublish.setEnabled(true);  
            }  
        });  
        ...  
    }  
    public static void main(String[] args) {  
        new NotebookServicePublish();  
    }  
}
```

Classe *NotebookServiceSOAPPublishUI.java* du
projet **jaxws-notebookwebservice**



Utilisée pour publier le
service web

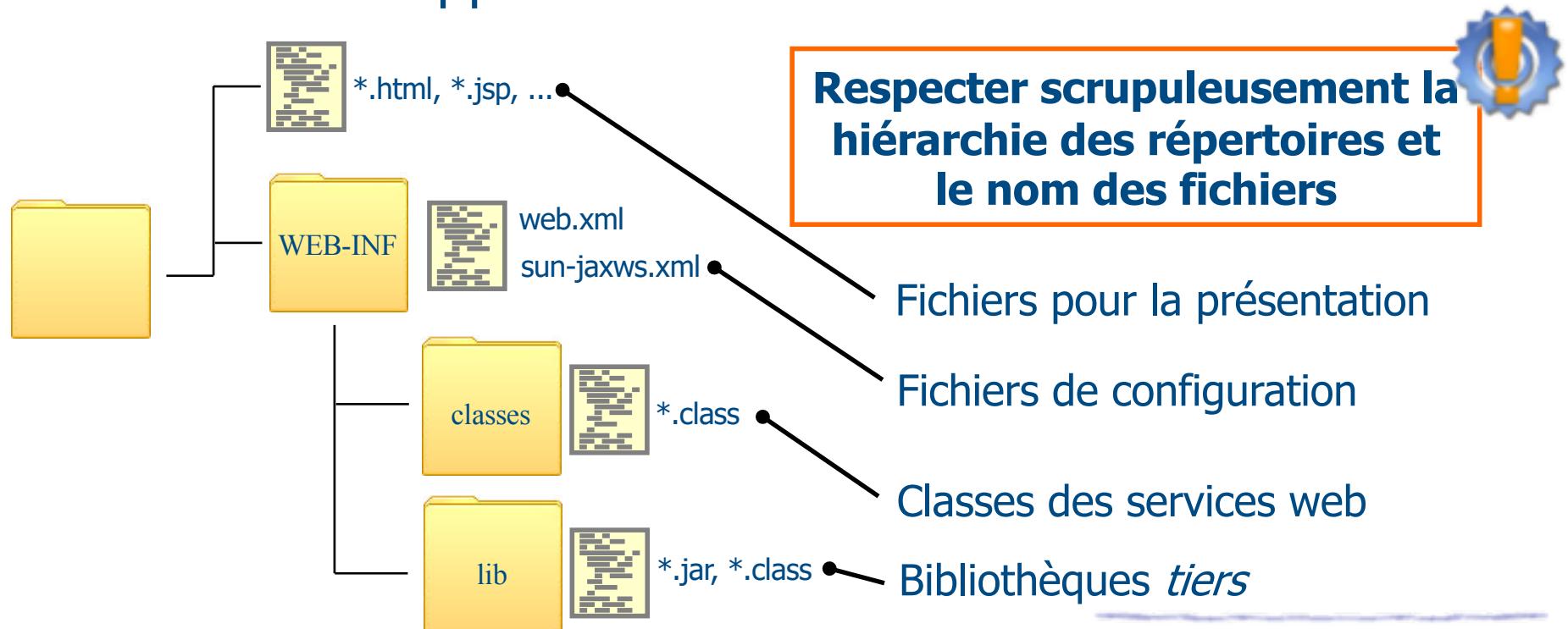
Utilisée arrêter la publication
du service web

Déploiement sur un serveur d'application Java

- Un service web est déployé dans une application web (un service web par application web)
- Différentes catégories de serveur d'application pour gérer les services web avec JAX-WS
 - Conteneur respectant JSR 109 (Implementing Enterprise Web Services)
 - La gestion du service web est transparente et maintenue par le serveur d'application
 - Exemple : **Glassfish** 
 - <http://jcp.org/en/jsr/summary?id=109>
 - Conteneur nécessitant une gestion par Servlet
 - Nécessite une configuration explicite du service web
 - Exemple : **Tomcat** 
 - Note : un composant additionnel se propose de fournir le support JSR 109 (<http://tomcat.apache.org/tomcat-6.0-doc/extras.html>)

Déploiement sur un serveur d'application Java

- Dans le cas d'un conteneur dont la gestion du service web est gérée par une Servlet
 - *web.xml* : précise la Servlet assurant la gestion
 - *sun-jaxws.xml* : utilisé pour effectuer une relation entre le contexte de l'application et la classe du service web
- Structure d'une application web fournissant un service web



Déploiement sur un serveur d'application Java

➤ Exemple : Tomcat et le fichier de configuration *web.xml*

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app 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">
    <description>HelloWorld</description>
    <display-name>HelloWorld</display-name>
    <listener>
        <listener-class>com.sun.xml.ws.transport.http.servlet.WSServletContextListener</listener-class>
    </listener>
    <servlet>
        <description>JAX-WS endpoint - helloworld</description>
        <display-name>helloworld</display-name>
        <servlet-name>helloworld</servlet-name>
        <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>helloworld</servlet-name>
        <url-pattern>/helloworld</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>
</web-app>
```

La Servlet est accessible
via cette URL

Servlet assurant la gestion du
service web

web.xml du projet
jaxws-helloworldWebService

Déploiement sur un serveur d'application Java

➤ Exemple : Tomcat et le fichier de configuration *sun-jaxws.xml*

```
<?xml version="1.0" encoding="UTF-8"?>

<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
    <endpoint
        name='helloworld'
        implementation='fr.mickaelbaron.helloworldwebservice.HelloWorldServiceImpl'
        url-pattern='/helloworld'/>
</endpoints>
```

sun-jaxws.xml du projet
jaxws-helloworldwebservice

Relation entre la classe
implémentant le service web et
la Servlet assurant sa gestion

Le fichier *sun-jaxws.xml* est également
utilisé pour ajouter des informations
supplémentaires (wsdl, binding...)



Déploiement sur un serveur d'application Java

➤ Exemple : générer un JAR et un WAR via Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>fr.mickaelbaron</groupId>
    <artifactId>jaxws-helloworldwebservice</artifactId>
    <packaging>${project.packaging}</packaging>

    <profiles>
        <profile>
            <id>jar</id>
            <activation>
                <activeByDefault>true</activeByDefault>
            </activation>
            <properties>
                <project.packaging>jar</project.packaging>
            </properties>
        </profile>
        <profile>
            <id>war</id>
            <build>
                <finalName>helloworldwebservice</finalName>
            </build>
            <properties>
                <project.packaging>war</project.packaging>
            </properties>
        </profile>
    </profiles>
</project>
```

Utilisation d'une variable pour définir la valeur du *packaging* (JAR ou WAR)

En fonction du profil choisi la variable \${project.packaging} est modifié en jar ou war

pom.xml du projet
jaxws-helloworldwebservice



Déploiement sur un serveur d'application Java

- Exemple : déployer un service web (packagé war) dans une instance Tomcat via Docker

```
$ mvn clean package -P war # Compile et build le projet JAXWS-HelloWorldWebService en invoquant le profil war  
=> Fichier helloworldwebservice.war disponible dans le répertoire target/  
  
$ docker pull tomcat:9.0.12-jre10-slim # Télécharge la version 9 de Tomcat avec une JRE 10  
=> Image Docker disponible  
  
$ docker run --rm --name helloworldservice-tomcat -v $(pwd)/target/helloworldwebservice.war:/usr/  
    local/tomcat/webapps/helloworldwebservice.war -it -p 8080:8080 tomcat:9.0.12-jre10-slim  
=> Service web disponible à l'adresse http://localhost:8080/helloworldwebservice/helloworld
```



Service web avec les EJB : généralités

- À partir d'un EJB Session, il est possible de définir le POJO associé comme étant un service web
- Pour rappel, l'appel à un EJB Session passe obligatoirement par un client écrit en Java
- Les avantages à transformer en EJB Session en service web
 - Caractéristiques des EJBs (transactions, sécurité, scalabilité...)
 - Plus grande hétérogénéité, le client n'est pas forcément écrit en Java
 - Réutilisabilité du code
 - Modularité (avec les annotations JAX-WS possibilité de masquer les méthodes qui ne doivent pas être découvertes)
- Nécessite d'avoir un conteneur EJB pour le déploiement
 - Glassfish, TomEE...

Service web avec les EJB : serveur

- Partir d'une classe Java définissant un EJB Session (*statefull* ou *stateless*)
- Ajouter l'annotation *@WebService*
- Déployer l'application sur un serveur d'application ayant le support EJB (Glassfish par exemple)
- Le conteneur EJB s'occupe de la gestion du service web, aucune Servlet n'est nécessaire
- Le document WSDL est généré automatiquement en respectant les valeurs par défauts
 - URL du WSDL : *http://monserveur/app/Service?WSDL*
- Toutes les méthodes de l'EJB sont par défaut des opérations du service web

Service web avec les EJB : serveur

➤ Exemple : définir un EJB Stateless comme étant un service web

Seule l'annotation `@Stateless` a été ajoutée

```
@Stateless
@WebService(endpointInterface = "fr.mickaelbaron.notebookwebservicefromejb.NotebookService")
public class NotebookServiceImpl {
    public boolean addPersonWithComplexType(Person newPerson) {
        ...
        return true;
    }

    public Person getPersonByName(String name) {
        Person current = new Person();
        current.setName(name);
        current.setBirthyear("1976");
        current.setAddress("17 Square Mickael BARON");
        return current;
    }

    public List<Person> getPersons() {
        Person first = new Person();
        ...
        Person second = new Person();
        ...

        List<Person> tabPerson = new ArrayList<Person>();
        tabPerson.add(first);
        tabPerson.add(second);
        return tabPerson;
    }

    public void addPersonWithSimpleType(String name, String address, String birthyear) {
        System.out.println("Name : " + name + " Address : " + address + " birthyear : " + birthyear);
    }
}
```

Le reste du code est le même que dans les exemples précédents



Service web avec les EJB : client

- Le développement du client est similaire au client développé précédemment où le point de départ est le document WSDL (via une URL ou via un fichier physique)
- Utilisation explicite de l'outil *wsimport* pour la génération du squelette du service web
 - Génération des classes liées à JAXB
 - Génération de classes service web (*PortType* et *Service*)
- Injecter un *@WebServiceReference* pour récupérer une instance du Service à partir du conteneur EJB
- Récupération d'un port via *get<ServiceName>Port()*
- Invocation des opérations

Service web avec les EJB : client

➤ Exemple : appel d'un service web à partir d'une Servlet

```
public class NotebookWebServiceFromEJBClientServlet extends HttpServlet {  
  
    @WebServiceRef(wsdlLocation = "http://localhost:8080/NotebookWebServiceFromEJB/Notebook?wsdl")  
    private Notebook_Service service;  
  
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
        response.setContentType("text/html;charset=UTF-8");  
        PrintWriter out = response.getWriter();  
        try {  
            out.println("<html>");  
            out.println("<head>");  
            out.println("<title>Servlet NotebookWebServiceFromEJBClientServlet</title>");  
            out.println("</head>");  
            out.println("<body>");  
            out.println("<h1>Servlet NotebookWebServiceFromEJBClientServlet at " + request.getContextPath() + "</h1>");  
  
            try {  
                Notebook port = service.getNoteBookPort();  
                Person newPerson = new Person();  
                newPerson.setName("BARON Mickael");  
                newPerson.setAddress("Poitiers");  
                newPerson.setBirthyear("1976");  
                boolean result = port.addPersonWithComplexType(newPerson);  
                out.println("<div>Result = " + result + "</div>");  
            } catch (Exception ex) {  
                ex.printStackTrace();  
            }  
            out.println("</body>");  
            out.println("</html>");  
        } finally {  
            out.close();  
        }  
    }  
}
```

La référence à
Notebook_Service est donnée
par le conteneur EJB

JAX-WS « in actions » : services web eBay

- eBay fournit via *eBay Developers Program* un accès à certains services via des services web étendus (legacy)
 - <http://developer.ebay.com/>
 - <https://developer.ebay.com/Devzone/finding/Concepts/FindingAPIGuide.html>
 - <https://developer.ebay.com/Devzone/shopping/docs/Concepts/ShoppingAPIGuide.html>
 - ...
- L'utilisation de ces services web est soumis à une inscription pour obtenir une clé d'autorisation
 - <https://developer.ebay.com/DevZone/account/>
- Nous montrons à partir de JAX-WS comment générer un client d'accès au service de recherche d'eBay (Finding API)
- Le code du client est fortement inspiré de
 - https://github.com/eBayDeveloper/eBay_APICall_CodeSamples

JAX-WS « in actions » : services web eBay

- Utilisation de l'outil *wsimport* via Maven
 - <http://developer.ebay.com/webservices/finding/latest/FindingService.wsdl>
- Génération des artifacts du service web
 - Génération des classes liées à JAXB (72 classes)
 - Génération de classes service web (2 classes)
- Création d'une instance de la classe *FindingService*
- Récupération d'un port *getFindingServiceSOAPPort()*
- Renseigner dans l'en-tête HTTP les informations de protocole et d'autorisation
- Préparer le paramètre de l'opération *findItemsAdvanced*
- Invocation de l'opération

JAX-WS « in actions » : services web eBay

➤ Exemple : création d'un client pour les Services Web eBay

```
public class EBAYFindingWebServiceClient {  
    private static final String CALLNAME = "findItemsAdvanced";  
    private static final String BASEURL = "http://svcs.ebay.com/services/search/FindingService/v1";  
    private static final String CATEGORY_ID = "267"; // Categorie pour Livres, BD et revues  
    private static final String KEYWORDS = "Game of thrones";  
    private static final String APPID = "TO_COMPLETE"; ●  
  
    public static void main(String[] args) {  
        // Initialization  
        FindingService service = new FindingService();  
        FindingServicePortType port = service.getFindingServiceSOAPPort();  
  
        BindingProvider bp = (BindingProvider) port;  
        Map<String, Object> requestProperties = bp.getRequestContext();  
        Map<String, List<String>> httpHeaders = new HashMap<String, List<String>>();  
        // Set the headers  
        httpHeaders.put("X-EBAY-SOA-OPERATION-NAME", Collections.singletonList(CALLNAME));  
        httpHeaders.put("X-EBAY-SOA-SECURITY-APPNAME", Collections.singletonList(APPID));  
        requestProperties.put(MessageContext.HTTP_REQUEST_HEADERS, httpHeaders);  
        requestProperties.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, BASEURL);  
  
        ... Suite dans le prochain transparent  
    }  
}
```

Modifier par votre identifiant de sécurité

Classe *EBAYFindingWebServiceClient* du projet
jaxws-ebayfindingwebserviceclient



JAX-WS « in actions » : services web eBay

➤ Exemple (suite) : création d'un client pour service web eBay

```
public class EBAYFindingWebServiceClient {  
    public static void main (String[] args) {  
        ...  
        // Prepare query.  
        FindItemsAdvancedRequest req = new FindItemsAdvancedRequest();  
        List<OutputSelectorType> opSelector = req.getOutputSelector();  
        opSelector.add(OutputSelectorType.SELLER_INFO);  
  
        ItemFilter objFilter1 = new ItemFilter();  
        objFilter1.setName(ItemFilterType.AVAILABLE_TO);  
        objFilter1.getValue().add("FR");  
        ...  
        List<ItemFilter> itemFilter = req.getItemFilter();  
        itemFilter.add(objFilter1);  
        itemFilter.add(objFilter2);  
        itemFilter.add(objFilter3);  
  
        List<String> catID = req.getCategoryId();  
        catID.add(CATEGORY_ID);  
        req.setSortOrder(SortOrderType.END_TIME_SOONEST);  
        req.setKeywords(KEYWORDS);  
  
        FindItemsAdvancedResponse res = port.findItemsAdvanced(req);  
  
        // Suite dans le prochain transparent  
    }  
}
```

Création du message d'entrée
à passer à l'opération
findItemsAdvanced(...)

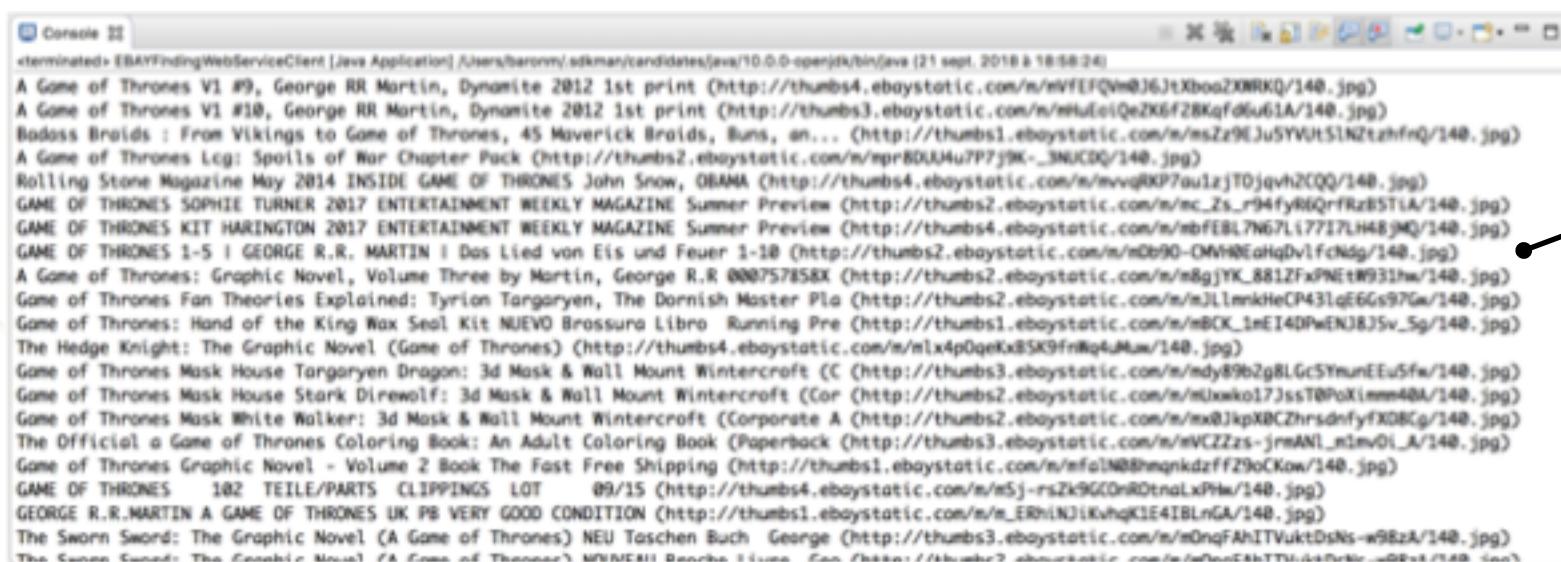
Appel de l'opération
permettant de rechercher

JAX-WS « in actions » : services web eBay

➤ Exemple (suite) : création d'un client pour service web eBay

```
public class EBAYFindingserviceClient {
    public static void main (String[] args) {
        ...
        // Extract result.
        SearchResult searchResult = res.getSearchResult();
        List<SearchItem> item = searchResult.getItem();
        for (SearchItem searchItem : item) {
            System.out.println(searchItem.getTitle() + " (" + searchItem.getGalleryURL() + ")");
        }
        System.out.println("Search Count: " + searchResult.getCount());
    }
}
```

Récupération
des résultats



```
Console [terminated] > EBAYFindingserviceClient [Java Application] /Users/baronm/sdkman/candidates/java/10.0.0-openjdk/bin/java (21 sept. 2018 à 18:58:24)
A Game of Thrones V1 #9, George RR Martin, Dynamite 2012 1st print (http://thumbs4.ebaystatic.com/m/mVFQVm8J6JtXbosZX0RKQ/148.jpg)
A Game of Thrones V1 #10, George RR Martin, Dynamite 2012 1st print (http://thumbs3.ebaystatic.com/m/mHuUe1QeZK6f28KqFd6u61A/148.jpg)
Bodoss Braids : From Vikings to Game of Thrones, 45 Maverick Braids, Buns, an... (http://thumbs1.ebaystatic.com/m/ms2z96Ju5YVUHS1NztzhfnQ/148.jpg)
A Game of Thrones Lcg: Spoils of War Chapter Pack (http://thumbs2.ebaystatic.com/m/nprBDUHU7P7j9k-3NUCDQ/148.jpg)
Rolling Stone Magazine May 2014 INSIDE GAME OF THRONES John Snow, OBAMA (http://thumbs4.ebaystatic.com/m/mvqRKP7ou1zjT0jqvh2CQ/148.jpg)
GAME OF THRONES SOPHIE TURNER 2017 ENTERTAINMENT WEEKLY MAGAZINE Summer Preview (http://thumbs2.ebaystatic.com/m/mc_2s_r94fyRIQrfRz85TLA/148.jpg)
GAME OF THRONES KIT HARINGTON 2017 ENTERTAINMENT WEEKLY MAGAZINE Summer Preview (http://thumbs4.ebaystatic.com/m/mfEBL7Nb7L777I7LH48jQ/148.jpg)
GAME OF THRONES 1-5 | GEORGE R.R. MARTIN | Das Lied von Eis und Feuer 1-10 (http://thumbs2.ebaystatic.com/m/m0b90-CMVHREahQdVlfcNdg/148.jpg)
A Game of Thrones: Graphic Novel, Volume Three by Martin, George R.R. 0008757858X (http://thumbs2.ebaystatic.com/m/m8gjYK_881ZFxPNEtW931hw/148.jpg)
Game of Thrones Fan Theories Explained: Tyrion Targaryen, The Dornish Master Pia (http://thumbs2.ebaystatic.com/m/mJLmnkHeCP431qE6Gs97Gw/148.jpg)
Game of Thrones: Hand of the King Wax Seal Kit NUEVO Brossura Libro Running Pre (http://thumbs1.ebaystatic.com/m/mBCK_1nEI4DPwENJ8JSv_5g/148.jpg)
The Hedge Knight: The Graphic Novel (Game of Thrones) (http://thumbs4.ebaystatic.com/m/mlx4p0qeKx85K9fnRq4Mum/148.jpg)
Game of Thrones Mask House Targaryen Dragon: 3d Mask & Wall Mount Wintercroft C (http://thumbs3.ebaystatic.com/m/mdy8962g8LGcSYmunEEu5fw/148.jpg)
Game of Thrones Mask House Stark Direwolf: 3d Mask & Wall Mount Wintercroft (Cor (http://thumbs2.ebaystatic.com/m/mLxxko17JssTBPoXimmm48A/148.jpg)
Game of Thrones Mask White Walker: 3d Mask & Wall Mount Wintercroft (Corporate A (http://thumbs2.ebaystatic.com/m/mx8JkpXBCZhrsdnfyfX0BCg/148.jpg)
The Official a Game of Thrones Coloring Book: An Adult Coloring Book (Paperback (http://thumbs3.ebaystatic.com/m/mVCZZzs-jmMANL_n2mV01_A/148.jpg)
Game of Thrones Graphic Novel - Volume 2 Book The Fast Free Shipping (http://thumbs1.ebaystatic.com/m/mfc1N88hmqnkdzff29oCkow/148.jpg)
GAME OF THRONES 182 TEILE/PARTS CLIPPINGS LOT 09/15 (http://thumbs4.ebaystatic.com/m/m5j-rsZk9G0DrnDtnalxPhw/148.jpg)
GEORGE R.R.MARTIN A GAME OF THRONES UK PB VERY GOOD CONDITION (http://thumbs1.ebaystatic.com/m/m_ERhINJ1Kvhqk1E4IBLnGA/148.jpg)
The Sworn Sword: The Graphic Novel (A Game of Thrones) NEU Taschen Buch George (http://thumbs3.ebaystatic.com/m/mDnqFAhITVuktDsNs-w9BzA/148.jpg)
The Seven Church: The Founder Novel TA Game of Thrones UNOFFICIAL Graphic Novel (http://thumbs2.ebaystatic.com/m/mfBnqk8TTU1247nRr...0000471AB.../148.jpg)
```

Une liste de
liens d'aperçu
d'image est
générée par
rapport à la
recherche
effectuée

