Middleware – Web Service – GMAIL - MAVEN

12 février 2018

1

Création d'un service Web Gmail

Analyse d'un service mail existant

1. Vous allez étudier un `service web` dédié aux mails comme celui déjà existant de `Yahoo`. Il vous servira d'exemple pour savoir créer votre propre service Mail. En effet, le service `YahooMail` contient énormément d'information sur le fonctionnement interne de leur service Web. (http://www.yahooapis.com/mail)

2. Par conséquent, vous allez créer votre propre service mail en *web servicisant* un serveur de mail déjà existant comme `gmail` par exemple. Dans ce contexte vous allez vous créer une adresse email sur gmail. (http://gmail.com)

Base revision 15fafec, Mon Jan 8 11:05:06 2018 +0700, David Bromberg.

3. L'interaction avec `gmail` se fera en `SMTP` et en `IMAP4`. Bien entendu, vous n'allez pas vous-même développer la pile protocolaire correspondante à ces protocoles. Vous aller utiliser l'`API JavaMail` de Sun. L'analyse de la documentation de son `API` vous sera utile par la suite. (http://www.oracle.com/technetwork/java/javamail/index.html)

Gestion du projet

On souhaite créer un nouveau projet `Maven` ayant comme particularité d'être un projet multi-module. Cela signifie que le projet va se décomposer d'une part d'une partie dédiée à la conception du code du client, et d'autre part d'une partie dédiée à la conception du code du serveur.

1. Créer donc un projet `Maven` ayant comme **groupID** `net.mail` et comme **artifactID** `gmail`. Par ailleurs, pour indiquer que l'on souhaite créer un projet hébergeant plusieurs sous-projets, on va utiliser un modèle de projet particulier ayant comme **archetypeGroupId** `org.codehaus.mojo.archetypes`, et ayant comme **archetypeArtifactId** `pom-root`.

```
# mvn archetype:generate
-DgroupId=net.mail
-DartifactId=gmail
-DarchetypeGroupId=org.codehaus.mojo.archetypes
-DarchetypeArtifactId=pom-root
```

2. Placez vous dans le répertoire de votre projet (c-a-d `gmail`). Créer de nouveau un projet dédié à votre service de mail.

```
 mvn archetype:generate
-DarchetypeArtifactId=maven-archetype-webapp
-DgroupId=net.mail.ws.service
-DartifactId=service
-Dversion=1.0-SNAPSHOT
```

On obtient alors la hiérarchie de répertoire suivante :

```
/
├── pom.xml
└── service
    ├── pom.xml
    └── src
        ├── main
        ├── resources
        └── webapp
            ├── WEB-INF
            │   └── web.xml
            └── index.jsp
```

3. Placez vous dans de nouveau à la racine de votre projet (c-a-d `gmail`). Créer, cette fois ci, un projet dédié au dévelopement du client, consommateur de votre service de mail.

```
mvn archetype:generate
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=net.mail.ws.client
-DartifactId=client
-Dversion=1.0-SNAPSHOT
```

On obtient alors la hiérarche de répertoires suivante :

```
gmail
├── pom.xml
├── client
│   ├── pom.xml
│   └── src
│       └── main
│           └── java
│               └── ...
└── service
    ├── pom.xml
    └── src
        ├── main
        ├── resources
        └── webapp
            ├── WEB-INF
            │   └── web.xml
            └── index.jsp
```

Si chaque sous-projet utilise une dépendance commune, comme le `framework Metro`, chacun doit gérer de façon locale, indépendemment les des autres, le numéro de version de cette dépendance. Ainsi, étant donné que les numéros de version sont répartis sur toute l'arborescence de votre projet, vous allez devoir éditer à la main chacun des fichiers `pom.xml` qui fait référence à cette dépendance pour être sûr de mettre à jour ce numéro de version dans chacun de vos sous projets. Afin de limiter ce problème, `Maven` fournit un moyen de gérer de façon centraliser le numéro de version correspondant aux différentes dépendances des sous-projets. Cet objectif est atteint *via* l'utilisation de la balise `<dependencyManagement><dependencies>...</dependencies></dependencyManagement>`. Enfin, les différents sous-projets peuvent avoir à utiliser un plugin en commun, comme par example dans notre cas `jaxws-maven-plugin`. Il est également possible de gérer de façon centralisée la dépendance vis-a-vis de `plugin` *via* l'utilisation de la balise `<pluginManagement>...</pluginManagement>`.

4. Modifier le fichier `pom.xml` situé à la racine de votre projet afin d'ajouter les dépendances communes de vos sous projet au sein des balises `<dependencyManagement>...</dependencyManagement>` et `<pluginManagement>...</pluginManagement>`.

Conception du service web mail

5. **Placez vous dans le sous-projet** `service`. Vous allez créer un service Web selon la méthode last contract. Cette méthode vous évite d'avoir à écrire vous même l'interface WSDL de votre service. Cela vous permet dans un premier temps de vous familiariser avec les documents WSDL, et XML. L'interface du service, c'est à dire son document WSDL associé, est généré automatiquement à partir d'une interface Java annotée qui représente votre service Web WSMail. Pour connaître la signification approfondie de toutes les annotations disponibles, vous pouvez consulter la documentation suivante. http://jax-ws.java.net/nonav/2.1.3/docs/annotations.html

6. Créer une interface Java `WSMail` définissant une méthode `sendMail` permettant d'envoyer un mail.

Listing 1– WSMail.java - Interface du service WSMail download

```
Line 1  package net.mail.ws.service;
    -   import javax.jws.WebMethod;
    -   import javax.jws.WebParam;
    -   import javax.jws.WebResult;
    5   import javax.jws.WebService;
    -   import javax.jws.soap.SOAPBinding;
    -
    -   @WebService(targetNamespace = "http://david.bromberg.fr/",
            name = "WSMail")
    -   @SOAPBinding(style = SOAPBinding.Style.RPC)
   10
    -   public interface WSMail {
    -
    -      @WebMethod
    -      public void sendMail(
   15            @WebParam(name = "smtpHost")
    -            java.lang.String smtpHost,
    -            @WebParam(name = "smtpPort")
    -            int smtpPort,
    -            @WebParam(name = "from")
   20            java.lang.String from,
    -            @WebParam(name = "to")
    -            java.lang.String to,
    -            @WebParam(name = "username")
    -            java.lang.String username,
   25            @WebParam(name = "password")
    -            java.lang.String password,
    -            @WebParam(name = "isSSL")
    -            boolean isSSL
    -      ) ;
   30   }
```

7. Dorénavant, vous allez implémenter le service par l'intermédiaire d'une classe Java `WSMailImpl` pour accéder *via* le protocole `SMTP` et `POP` à gmail en utilisant le `framework JavaMail`. Vous devez obtenir le fichier suivant. **Attention, l'exemple donné ne comprend qu'une seule opération, à savoir, envoyer un email. En effet, le fichier n'est donné qu'à titre d'exemple. Vous devez par la suite dans ce TD/TP créer vos propres méthodes de contrôle de compte mail.**

Listing 2– WSMailImpl.java - Implémentation du service WSMail download

```
Line 1  package net.mail.ws.service;
        -
        -  import java.lang.reflect.Array;
        -  import java.util.Collection;
        5  import java.util.Properties;
        -  import java.util.Vector;
        -
        -  import javax.mail.*;
        -  import javax.mail.internet.*;
       10  import javax.mail.search.FromStringTerm;
        -  import javax.mail.search.SearchTerm;
        -  import javax.jws.WebParam;
        -  import javax.jws.WebService;;
        -
       15  @WebService(endpointInterface = "␣
               net.mail.ws.service.WSMail", serviceName = "
               MyOwnMailAgent")
        -
        -  public class WSMailImpl {
        -
        -      public void sendMail (String smtpHost, int smtpPort,
       20                            String from, String to, String
                                         username, String password,
                                         boolean isSSL){
        -          Properties props = new Properties();
        -
        -          if(isSSL) {
        -              props.put("mail.transport.protocol", "smtps");
       25              props.put("mail.smtps.auth", "true");
        -          } else {
        -              props.put("mail.transport.protocol", "smtp");
        -              props.put("mail.smtps.auth", "false");
        -          }
       30
        -          props.put("mail.smtp.host", smtpHost);
        -
```

```
        Session session = Session.getDefaultInstance(props)
            ;
        session.setDebug(true);

        try {
            MimeMessage message = new MimeMessage(session);
            message.setFrom(new InternetAddress(from));
            message.addRecipient(Message.RecipientType.TO,
                new InternetAddress(to));
            message.setSubject("Hello");
            message.setText("Hello World");

            Transport tr;
            tr = session.getTransport();
            tr.connect(smtpHost, smtpPort, username,
                password);
            message.saveChanges();

            tr.sendMessage(message,message.getAllRecipients
                ());
            tr.close();
        }
        catch (Exception e) {
            System.out.println("MessagingException");
        }
    }
}
```

8. On s'intéresse maintenant à la compilation du service. En plus de la dépendance avec le *framework* `jax-ws` pour créer un `web service`, il existe dorénavant une dépendance forte avec la librairie `JavaMail`. Par conséquent, il faut indiquer à Maven cette nouvelle dépendance. De ce fait, ajouter la dépendance suivante au fichier pom.xml du projet service.

Listing 3– pom.xml

```
...
<dependency>
 <groupId>javax.mail</groupId>
 <artifactId>mail</artifactId>
 <version>1.4</version>
</dependency>
...
```

9. L'étape suivante est de générer le document WSDL correspondant à l'interface Java WSMail. Pour cela, on utilise l'outil `wsgen` disponible *via* le plugin `jaxws-maven-plugin`. Modifier en conséquence le fichier pom.xml de votre projet.

10. Saisir la commande `mvn clean install`. Vérifier le document WSDL généré dans le répertoire target/jaxws/wsgen/wsdl/. Le service est créé ;-).

11. Déployer le service web mail en utilisant `jetty`. Modifier en conséquence votre fichier pom.xml.

2

Invocation du service

1. Placez vous dans le sous projet `client`. Pour coder le client nous allons bien évidemment réutiliser le document WSDL de notre service accessible à l'url `http://localhost:8080/WSmail/services/mail?wsdl`. Comme vu dans le précédent TD/TP, vous allez utiliser l'utilitaire wsimport pour générer les stub(s) nécessaire pour invoquer votre service distant WSMail. Configurer le fichier pom.xml de façon à ajouter l'exécution de l'action wsimport.

2. Suivant le code Java généré vous devriez être en mesure de coder le client. Voici un exemple de code source possible.

Listing 4– WSMailClient.java - Implémentation du client

```
Line 1  package net.mail.ws.client;
-
-   public class WSMailClient {
-       public static void main (String[] args) {
5
-           MyOwnMailAgent service =new MyOwnMailAgent();
-           WSMail mail = service.getWSMailImplPort();
-           mail.sendMail("smtpHost", smtpPort,"from", "to","
                login","passwd", "isSSL");
-       }
10  }
```

   ➠ Pour envoyer un mail via un compte `google`, utiliser l'hôte suivant **smtp.gmail.com** avec le **port 465**, et activer le **cryptage**.

3. Executer le client et vérifier votre compte mail

4. Maintenant que vous avez testé le bon fonctionnement de la première fonction de votre service mail, implémenter au choix une ou plusieurs des fonctions manquantes en vous inspirant du service Yahoo. Vous pouvez par exemple ajouter les fonctions suivantes :

- Envoie de mails avec pièces jointes.
- Récupération de mails.
- Déplacement de mails.
- Création de répertoire.
- Suppression de mails en fonction de plusieurs critères.
- Recherche de mails multicritères (selon le sujet, l'émetteur, la date etc.).
- Une méthode d'authentification évitant de s'authentifier à chaque requête.

Pour chaque fonction ajoutée, n'oubliez pas de changer l'interface de votre service.

5. Rajouter la gestion des exceptions dans votre code, ainsi que la gestion de type complexe en tant que type pour les paramètres passés en argument et la valeur de retour. Comment cela se traduit-il dans le fichier WSDL ?