目次

改訂情報

変更年月日	変更内容
2013-10-01	初版
2014-04-01	第２版　下記を追加・変更しました ■ 「*Webサービス・クライアント を実装する*」 にコラムを追記しました。
2017-04-01	第３版　下記を追加・変更しました ■ 「*Webサービス・クライアント の作成*」 の「Webサービス用ツール」のダウンロード方法を修正しました。

はじめに

本書の目的

本書では intra-mart Accel Platform における Webサービス を、Java言語 によって作成、提供する方法について説明します。

説明範囲は以下のとおりです。

- Webサービス・プロバイダ の作成方法とサンプルの解説
- Webサービス・クライアント の作成方法とサンプルの解説

対象読者

本書では次の利用者を対象としています。

- intra-mart Accel Platform の Webサービス を理解している
- Webサービス・プロバイダ の提供を行うアプリケーションの開発者
- Webサービス・クライアント を利用したアプリケーションの開発者

注意事項

1. Webサービス を利用するにあたり、いくつかの制限事項が存在します。制限事項についての詳細は「リリースノート」-「Webサービスの制限事項」を参照してください。
2. 本書で解説するチュートリアルの資材は、一部「Webサービス スクリプト開発プログラミングガイド」と重複しています。
   本書で作成した資材と「Webサービス スクリプト開発プログラミングガイド」で作成した資材を同時にデプロイすると、サンプルが動作しないことがあります。

本書の構成

- *概要*

  Javaにおける Webサービス の概要情報について説明します。

- *Webサービス・プロバイダ の作成*

  Javaを利用して、intra-mart Accel Platform 上に Webサービス をデプロイする方法について説明します。

- *Webサービス・クライアント の作成*

  Javaを利用して、intra-mart Accel Platform 上にデプロイされた Webサービス を利用する方法について説明します。

- *付録*

  Webサービス を開発する上で発生する問題と解決方法や、追加のサンプルコードなどを収録しています。

概要

項目

Webサービスとは

本書では、「Webサービス」とは SOAP と WSDL を用いた Webサービス を指します。
Webサービス の詳細については「Webサービス 認証・認可 仕様書」を参照してください。

本書のチュートリアルを進める上での注意点

- チュートリアルでは e Builder を使用します。

  チュートリアルで作成する資材を開発・デプロイするために、e Builder を使用します。
  あらかじめご用意ください。

- 一般的なJavaの知識が必要です。

  Webサービス・プロバイダ および Webサービス・クライアント を作成するため一般的なJavaの知識が必要です。

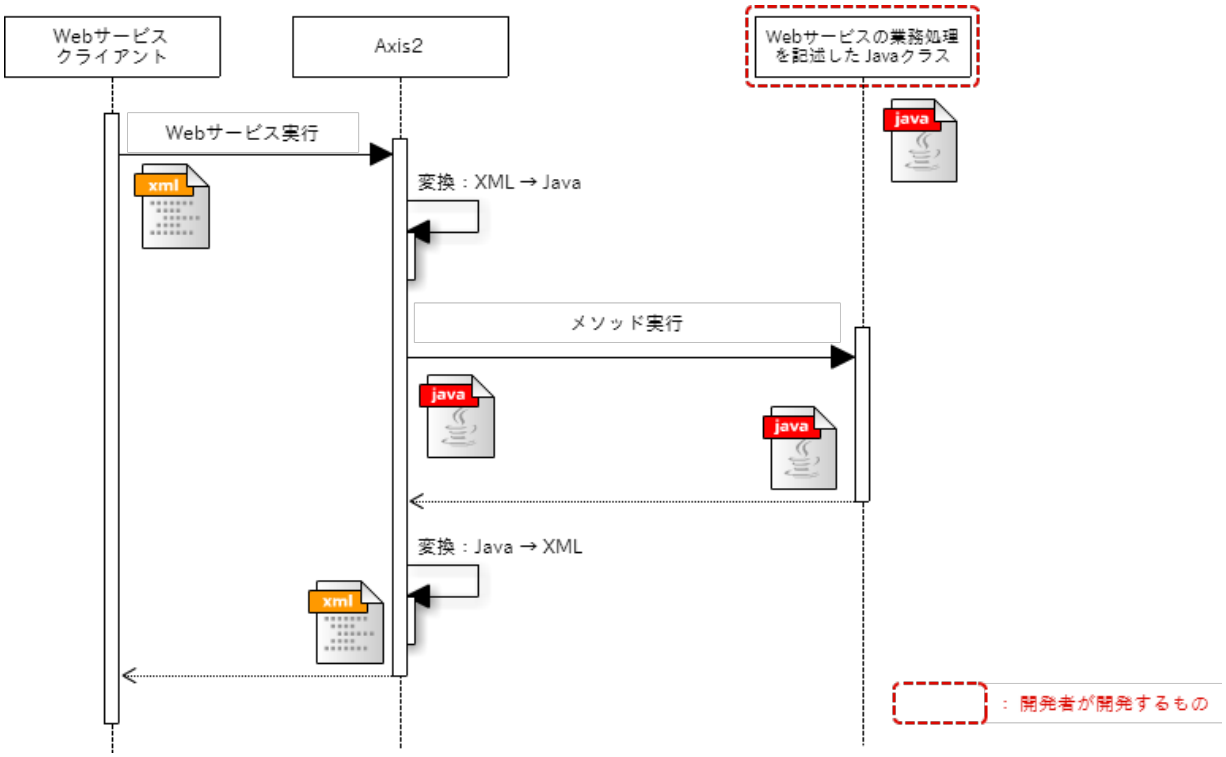
Webサービス・プロバイダ の作成

項目

作成手順の概要

この章では、Webサービス として公開する手順を説明します。
Webサービス として公開するJavaクラスを作成します。

Webサービス として公開するJavaクラスが実行されるまでの流れは以下の通りです。



1. Webサービス・クライアント が、Webサービス の実行を要求します。
2. Web サービス実行エンジン「Apache Axis2」が、受け付けたリクエストに該当するJavaクラスのメソッドを呼び出します。
3. 実行結果を Webサービス・クライアント に返却します。

Webサービス のデプロイを準備する

開発環境を用意する

最初に Web サービスを提供するための資材を作成する開発環境を用意します。
このチュートリアルでは e Builder を使用して、以下のプロジェクトを作成し、開発を行う手順を説明します。

グループID	mypackage （デフォルトの設定を使用）
バージョン	1.0.0 （デフォルトの設定を使用）
プロジェクト名	sample_provider

まず、e Builder、Resin、および、intra-mart Accel Platform をインストールして開発環境を構築してください。
インストール手順は「intra-mart e Builder for Accel Platform セットアップガイド」を参照してください。

> **注意**
>
> intra-mart Accel Platform をインストールする際、ベースモジュールから「Webサービス 認証・認可」と「Webサービス 認証・認可クライアント」モジュールを選択してください。
> 選択しない場合、チュートリアルの Java コードでコンパイルエラーが発生します。

e Builder のインストールが完了したら、以下の手順に従って、「Module Project」でプロジェクトを作成します。
詳しくは「intra-mart e Builder for Accel Platform アプリケーション開発ガイド」-「モジュール・プロジェクト作成」を参照してください。

1. 「ファイル」ー「新規」ー「プロジェクト」をクリックします。
2. 「e Builder」ー「Module Project」を選択して「次へ」をクリックします。
3. プロジェクト名に「sample_provider」を入力して、「終了」をクリックします。

プロジェクトの作成が完了したら、intra-mart Accel Platform の API を使用できるようにするために、プロジェクトの設定を行います。
詳しくは「intra-mart e Builder for Accel Platform アプリケーション開発ガイド」-「プロジェクトの設定」を参照してください。

1. プロジェクトを右クリックして「プロパティ」を選択します。
2. 「e Builder」ー「Module Assembly」を選択します。
3. Web アーカイブディレクトリに、war を展開してできたコンテキストパスと同名のフォルダを選択します。
4. リソース変更時の自動デプロイ先の一覧で、全てのチェックボックスを外します。
5. 「OK」をクリックします。

依存関係を解決する

プロジェクトの設定が完了したら、依存関係の修正を行います。
Webサービス・プロバイダ を作成するためには、「Webサービス 認証・認可」モジュールに依存する必要があります。

以下の手順に従って、プロジェクトの依存関係を修正します。

1. 作成したプロジェクトのルートディレクトリに配置されている「module.xml」をダブルクリックします。
2. 「依存関係」タブを開き、「追加」をクリックします。
3. 以下の内容を入力して、「OK」をクリックします。

ID	jp.co.intra_mart.im_ws_auth
バージョン	8.0.2

> **コラム**
>
> 基本的にバージョンはサポートが行われている番号を指定します。
> 使用したい API が他のバージョンに含まれている場合、そのバージョン番号を指定してください。

4. module.xml ファイルを保存した後、「module.xml」タブを開き、不要なタグ（<tags>）を除去します。
   最終的に以下のようなソースに修正します。

```
<?xml version="1.0" encoding="UTF-8"?>
<module conf:schemaLocation="urn:intramart:jackling:toolkit:configurations configurations.xsd"
      xmlns="urn:intramart:jackling:module" xmlns:conf="urn:intramart:jackling:toolkit:configurations"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemeLocation="urn:intramart:jackling:module module.xsd">

   <id>mypackage.sample_provider</id>
   <version>1.0.0</version>
   <type>module</type>
   <name>${module.name}</name>
   <vendor>${module.vendor}</vendor>
   <description>${module.description}</description>

   <dependencies>
      <dependency>
         <module-id>jp.co.intra_mart.im_ws_auth</module-id>
         <verified-version min="8.0.2">8.0.2</verified-version>
      </dependency>
   </dependencies>
</module>
```

5. 「依存関係の階層」タブを開き、依存関係が解決されさまざまなモジュールが表示されていれば成功です。

Javaで業務処理を作成する

プロジェクトの準備が完了したら、Webサービス として公開するJavaクラスを作成し、業務処理を記述します。

今回作成するサンプルのWeb サービスでは、以下の形式のメンバ情報を保存・検索します。

プロパティ	説明（型）
id	メンバID（String）
name	メンバ名（String）
age	年齢（Integer）
married	既婚の場合 true、未婚の場合 false（Boolean）
birthDate	生年月日（Date）
children	子供情報（メンバ情報形式の配列）

型情報クラスを作成する

Web サービスで取り扱うJavaBeansを作成します。

以下の手順に従って、e Builder で型情報クラスを作成します。

1. プロジェクトを右クリックして、「新規」ー「クラス」を選択します。
2. 以下の内容を入力して、「OK」をクリックします。

パッケージ	sample.web_service.provider
名前	Member

3. Member.java がプロジェクトの src/main/java 配下に作成されます。
4. メンバ情報の各プロパティを保持するための private 変数、および、アクセサメソッドを定義します。

```
package sample.web_service.provider;

import java.io.Serializable;
```

```
import java.util.Date;

public class Member implements Serializable {

   // 用途に応じて、変更する必要がある場合は変更してください。
   private static final long serialVersionUID = 1L;

   private String id;

   private String name;

   private Integer age;

   private Boolean married;

   private Date birthDate;

   private Member[] children;

   public Integer getAge() {
      return age;
   }

   public Date getBirthDate() {
      return birthDate;
   }

   public Member[] getChildren() {
      return children;
   }

   public String getId() {
      return id;
   }

   public Boolean getMarried() {
      return married;
   }

   public String getName() {
      return name;
   }

   public void setAge(final Integer age) {
      this.age = age;
   }

   public void setBirthDate(final Date birthDate) {
      this.birthDate = birthDate;
   }

   public void setChildren(final Member[] children) {
      this.children = children;
   }

   public void setId(final String id) {
      this.id = id;
   }

   public void setMarried(final Boolean married) {
      this.married = married;
   }

   public void setName(final String name) {
      this.name = name;
   }

}
```

コラム

当サンプルではメンバ情報をPermanent APIで扱うため、Member クラスは Serializable インタフェースを実装しています。

注意

Web サービスとして公開するメソッドの引数、および、返却値に、継承関係を持つクラスを使用しないでください。
継承関係を持ったクラスを使用すると、Web サービスのクライアント側でエラーが発生します。
これは、Java オブジェクトが XML に変換される際、XML 名前空間がサブクラスで統一される ADB（Axis Data Binding）の仕様による制限です。

例えば、以下の SubModel が ParentModel の子クラスとして定義されている場合、SubModel は Web サービスとして公開するメソッドの引数、および、返却値として使用できません。

- sample.foo.ParentModel
- sample.bar.SubModel

Web サービス処理を実装する

sample.web_service.provider.MemberInfoOperatorService.java を用意します。
このクラスに定義されているメソッド「add()」、「find()」、および、「findAll()」を Web サービスとして公開します。

以下の手順に従って、e Builder で Webサービス として公開するJavaクラスを作成します。

1. プロジェクトを右クリックして、「新規」ー「クラス」を選択します。
2. 以下の内容を入力して、「OK」をクリックします。

パッケージ	sample.web_service.provider
名前	MemberInfoOperatorService

3. MemberInfoOperatorService.java がプロジェクトの src/main/java 配下に作成されます。
4. MemberInfoOperatorService.java を実装します。
   MemberInfoOperatorService.java のソースは以下の通りです。

```
package sample.web_service.provider;

import java.io.IOException;
import java.util.List;

import jp.co.intra_mart.foundation.service.client.information.PermanentDirectory;
import jp.co.intra_mart.foundation.service.client.information.TreasureFile;
import jp.co.intra_mart.foundation.web_service.auth.WSUserInfo;

import org.apache.axis2.AxisFault;

public class MemberInfoOperatorService {

   private static final String DOMAIN = "sample_web_service";

   private static final String GROUP = "sample_member_info";

   public Boolean add(final WSUserInfo wsUserInfo, final Member member) throws AxisFault {
      final TreasureFile<Member> treasure = getPermanentFile();
      try {
         treasure.put(member.getId(), member);
         return Boolean.TRUE;
      } catch (final IOException e) {
         throw AxisFault.makeFault(e);
      } catch (final ClassNotFoundException e) {
         throw AxisFault.makeFault(e);
      }
   }

   public Member find(final WSUserInfo wsUserInfo, final String id) throws AxisFault {
      final TreasureFile<Member> treasure = getPermanentFile();
      try {
         final Member member = treasure.get(id);
         return member;
      } catch (final IOException e) {
         throw AxisFault.makeFault(e);
      } catch (final ClassNotFoundException e) {
         throw AxisFault.makeFault(e);
      }
   }

   public Member[] findAll(final WSUserInfo wsUserInfo) throws AxisFault {
      final TreasureFile<Member> treasure = getPermanentFile();
      try {
         final List<String> keyList = treasure.keyList();
         final int size = keyList.size();
         final Member[] members = new Member[size];
         for (int i = 0; i < size; i++) {
            members[i] = treasure.get(keyList.get(i));
         }
         return members;
      } catch (final IOException e) {
         throw AxisFault.makeFault(e);
      } catch (final ClassNotFoundException e) {
         throw AxisFault.makeFault(e);
      }
   }

   private TreasureFile<Member> getPermanentFile() {
      return PermanentDirectory.getInstance(DOMAIN).getFile(GROUP);
   }

}
```

コラム

メンバ情報の保存は Permanent API を利用します。

> **コラム**
>
> AxisFault をスローすることで、スローした内容を Webサービス・クライアント に返信できます。
> AxisFaultの詳細は、「Axis2のAPIドキュメント」を参照してください。

services.xml を作成する

Webサービス 実装クラスの作成が完了したら、Web サービスの設定ファイル「services.xml」を用意します。

<services.xml>ファイルを、プロジェクトの以下の場所に作成します。

- src/main/webapp/WEB-INF/services/sample_member_info/META-INF/services.xml

services.xml のソースは以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
  <service name="SampleMemberInfoOperatorService">
    <parameter name="ServiceClass">sample.web_service.provider.MemberInfoOperatorService</parameter>
    <module ref="im_ws_auth"/>
    <messageReceivers>
      <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
      <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </messageReceivers>

    <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>

    <operation name="add">
      <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>
    </operation>

    <operation name="find">
      <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>
    </operation>

    <operation name="findAll">
      <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>
    </operation>
  </service>
</serviceGroup>
```

> **コラム**
>
> 各関数に、アクセス権限を設定するための認可の「リソースURI」を設定します。
> 最初はサンプルの動作を確認するために、未認証・認証済みを問わず、どのユーザでもアクセス可能なリソースURIを割り当てます。
>
> - service://intra-mart.jp/public-resources/welcome-to-intramart
>
> 実際に Web サービスを業務利用する場合は、個別の「リソースURI」を用意してください。
> 権限設定の詳細は、後述の「*認可を利用してアクセス権限を設定する*」を参照してください。

Webサービス をデプロイする

資材をデプロイする

以上で Web サービスを提供するための資材が完成しました。
次に作成したモジュールをユーザモジュールとして取り込み、war を作成して Resin にデプロイします。

以下の手順に従って、e Builder でユーザモジュールを作成します。

1. プロジェクトを右クリックして、「エクスポート」を選択します。
2. 「e Builder」ー「imm file」を選択して、「次へ」をクリックします。
3. 出力先フォルダに任意の場所を選択して、「終了」をクリックします。
4. しばらくすると、出力先フォルダに <sample_provider-1.0.0.imm> ファイルが作成されます。

次に、以下の手順に従って、e Builder で imm ファイルをユーザモジュールとして取り込みます。

1. e Builder で環境構築時に利用したプロジェクトの「juggling.im」を開きます。
2. 「ユーザモジュール」タブを開き、右上の「モジュールを追加します」アイコンをクリックします。
3. e Builder で作成した <sample_provider-1.0.0.imm> ファイルを選択して開きます。

> **コラム**
>
> 依存関係が不足している場合、上側にエラーメッセージが表示されます。
> この場合、エラーメッセージをクリックして不足しているモジュールを追加してください。

4. juggling.im を保存して、環境構築時と同じ手順で war を作成し、Resin にデプロイします。
5. Resin を再起動します。

次に、以下の手順に従って、テナント環境セットアップを実施します。

1. システム管理画面を開き、システム管理者でログインします。
   `http://<HOST>:<PORT>/<CONTEXT_PATH>/system/login`
2. 「テナント環境セットアップ」をクリックします。

> **コラム**
>
> テナント環境が最新であり、セットアップが必要なモジュールがない旨のメッセージが表示されている場合は、以降の操作は不要です。

3. 続けて「テナント環境セットアップ」をクリックします。
4. 確認メッセージで「決定」をクリックします。

デプロイされていることを確認する

Resin の再起動とテナント環境セットアップが完了したら、作成した Web サービスがデプロイされていることを確認します。

以下の手順に従って、正常にデプロイされていることを確認します。

1. 以下の URL にアクセスします。
   `http://<HOST>:<PORT>/<CONTEXT_PATH>/services/listServices`
2. 「SampleMemberInfoOperatorService」に「Service Status : Active」の文字列と、各関数名が表示されていることが確認できれば成功です。

認可を利用してアクセス権限を設定する

ここまでのチュートリアルでは、 Web サービスに割り当てた認可のリソースURIは、未認証・認証済みを問わず、どのユーザでもアクセス可能なものを割り当てました。

- service://intra-mart.jp/public-resources/welcome-to-intramart

Web サービスを正式版として提供する際は、Web サービスのアクセス権限を細かく設定できるようにするために、認可リソースを登録します。
Web サービスの場合、認可リソースのキーとなる「リソースURI」のスキームは、通常「service」を利用します。
例えば、以下のようにリソースURI を定義します。

- service://sample_provider/web_service/member_info_operator

Web サービスの各関数に対して個別に権限設定を分けて管理したい場合、各関数にリソースURI を定義します。
例えば、サンプルの Web サービスで公開した「add()」、「find()」、「findAll()」のそれぞれで権限設定を分けたい場合は、３つのリソースを作成するためにそれぞれ「リソースURI」を定義します。

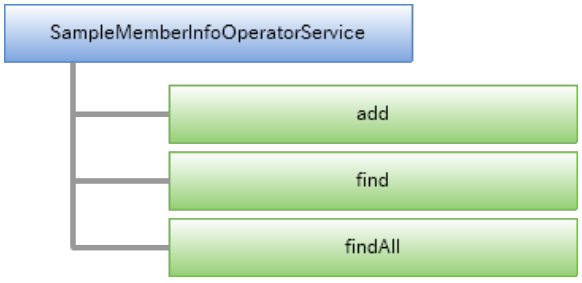
- service://sample_provider/web_service/member_info_operator/add
- service://sample_provider/web_service/member_info_operator/find
- service://sample_provider/web_service/member_info_operator/findAll

認可リソースを作成する

権限設定を行うための「リソースURI」が決定したら、テナント環境セットアップ資材を作成します。
必要な資材は以下の通りです。

- 認可リソース設定ファイル
- テナント環境セットアップを実施するためのセットアップ設定ファイル

この章では、認可に以下の構成でリソースを登録します。



> **コラム**
>
> 認可リソースは、テナント環境セットアップの資材からではなく、テナント管理機能の「認可設定画面」から登録することもできます。
> Web サービスの開発中に認可リソースを頻繁に変更する可能性がある場合は、認可設定画面から設定すると便利です。
> 認可設定画面を操作する方法についての詳細は「テナント管理者操作ガイド」-「認可を設定する」の「リソースを追加する」を参照してください。

それぞれ必要な資材を、プロジェクトの以下の場所に作成します。 サンプルの資材内容は以下の通りです。

- 認可リソース設定ファイル

  src/main/storage/system/products/import/basic/sample_provider/sample_provider-authz-resource.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.intra-mart.jp/authz/imex/resource">

  <authz-resource id="sample_provider-service"
uri="service://sample_provider/web_service/member_info_operator">
    <display-name>
      <name locale="ja">SampleMemberInfoOperatorService</name>
    </display-name>
    <parent-group id="web-services" />
  </authz-resource>

  <authz-resource uri="service://sample_provider/web_service/member_info_operator/add">
    <display-name>
      <name locale="ja">add</name>
    </display-name>
    <parent-group id="sample_provider-service" />
  </authz-resource>

  <authz-resource uri="service://sample_provider/web_service/member_info_operator/find">
    <display-name>
      <name locale="ja">find</name>
    </display-name>
    <parent-group id="sample_provider-service" />
  </authz-resource>

  <authz-resource uri="service://sample_provider/web_service/member_info_operator/findAll">
    <display-name>
      <name locale="ja">findAll</name>
    </display-name>
    <parent-group id="sample_provider-service" />
  </authz-resource>

</root>
```

> **コラム**
>
> Web サービス自身と、各関数に割り当てた「リソースURI」分、認可リソースを作成します。
>
> Web サービスに関係する認可リソースを登録するための親リソースグループ「web-services」が初期状態で用意されています。
> そのため、サンプルの Web サービスのトップ階層となる「SampleMemberInfoOperatorService」の親リソースグループを「web-services」に設定します。

- テナント環境セットアップを実施するためのセットアップ設定ファイル

  src/main/conf/products/import/basic/sample_provider/import-sample_provider-config-1.xml

```
<import-data-config xmlns="http://intra_mart.co.jp/system/service/provider/importer/config/import-data-config"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://intra_mart.co.jp/system/service/provider/importer/config/import-data-config import-data-config.xsd">

   <tenant-master>
      <authz-resource-file>products/import/basic/sample_provider/sample_provider-authz-resource.xml</authz-resource-file>
   </tenant-master>

</import-data-config>
```

services.xml を変更する

セットアップ資材の作成が完了したら、Web サービスの各関数にアクセスするための権限設定ファイル（services.xml）を変更します。

- src/main/webapp/WEB-INF/services/sample_member_info/META-INF/services.xml

service://intra-mart.jp/public-resources/welcome-to-intramart の部分を、定義した「リソースURI」に書き換えます。
services.xml のソースは以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
   <service name="SampleMemberInfoOperatorService">
      <parameter name="ServiceClass">sample.web_service.provider.MemberInfoOperatorService</parameter>
      <module ref="im_ws_auth"/>
      <messageReceivers>
         <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
         <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
      </messageReceivers>

      <parameter name="authz-uri">service://sample_provider/web_service/member_info_operator</parameter>

      <operation name="add">
         <parameter name="authz-uri">service://sample_provider/web_service/member_info_operator/add</parameter>
      </operation>

      <operation name="find">
         <parameter name="authz-uri">service://sample_provider/web_service/member_info_operator/find</parameter>
      </operation>

      <operation name="findAll">
         <parameter name="authz-uri">service://sample_provider/web_service/member_info_operator/findAll</parameter>
      </operation>
   </service>
</serviceGroup>
```

> **コラム**
>
> services.xml の設定内容については「Webサービス 認証・認可 仕様書」-「services.xmlについて」もあわせて参照してください。

資材を再デプロイする

Web サービスの設定ファイルの更新が完了したら、再デプロイを行います。
「*資材をデプロイする*」の手順に従って、 sample_provider-1.0.0.imm を再デプロイしてください。

> **注意**
>
> デプロイ後、システム管理画面から、テナント環境セットアップを必ず実行してください。

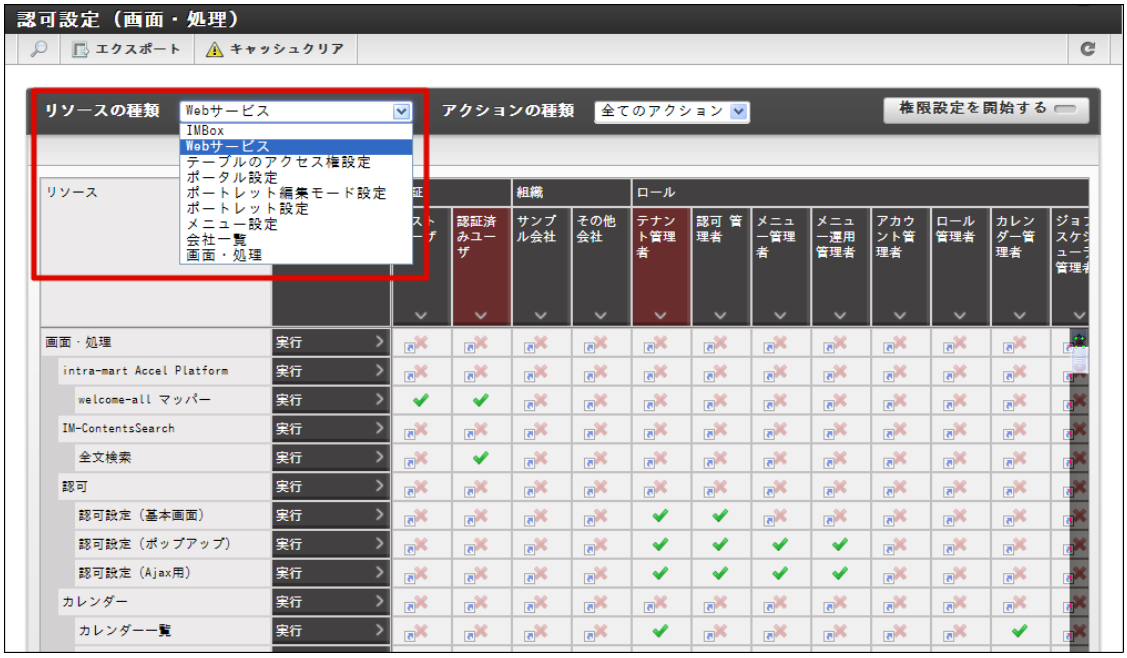
認可で権限設定を行う

再デプロイが完了したら、認可設定画面を開いて実際に Web サービスに対してアクセス権限を設定します。

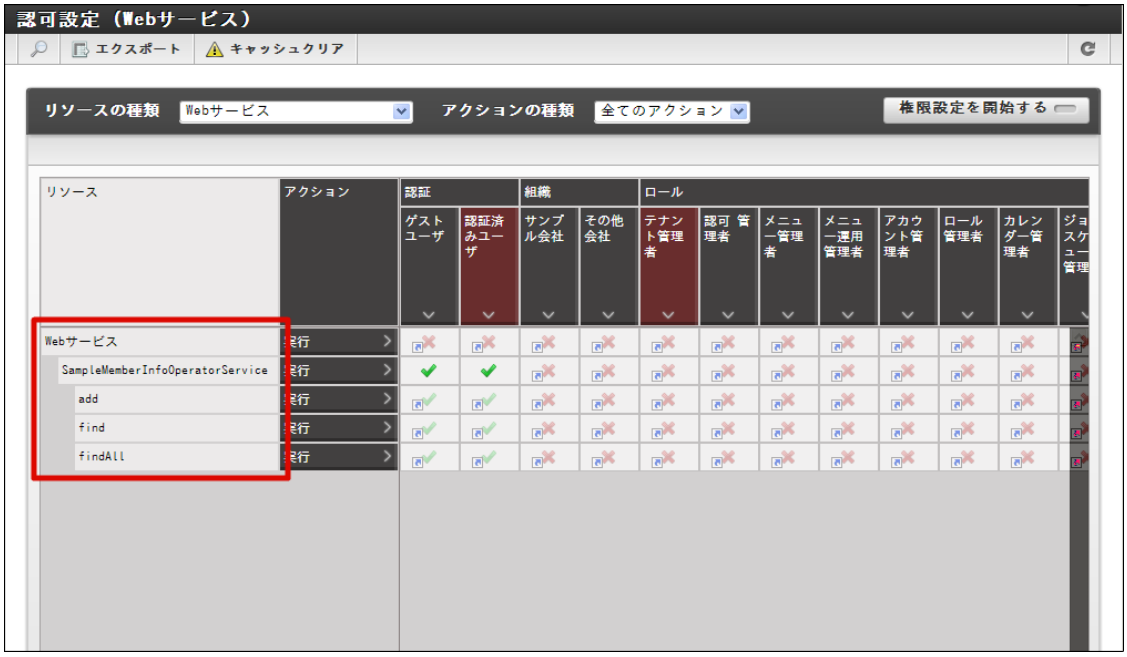
以下の手順に従って、アクセス権限を設定します。

1. 一般利用者のログイン画面を開き、テナント管理者でログインします。
   http://<HOST>:<PORT>/<CONTEXT_PATH>/login
2. サイトマップを開き、「テナント管理」カテゴリから「認可」をクリックします。

3. 「リソースの種類」から「Web サービス」を選択します。



4. 認可設定のグリッドに「SampleMemberInfoOperatorService」とその配下に「add」、「find」、および、「findAll」が表示されていることを確認します。



5. 「add」、「find」、および、「findAll」に対して、任意の対象者条件に権限を設定します。

> **コラム**
>
> 権限の設定方法についての詳細は「テナント管理者操作ガイド」-「認可を設定する」を参照してください。
>
> 実際に設定したアクセス権限通りに動作するかどうかを確認するためには、「Webサービス・クライアント」を用意する必要があります。

Webサービス・クライアント の作成

項目

作成手順の概要

この章では、Web サービスとして公開されたオペレーションを、Javaから利用する手順を説明します。
このサンプルでは、WSDL からスタブクラスを作成し Webサービス・クライアント を実装します。
スタブを利用することにより、XMLを意識することなく、Web サービスを呼び出すことができます。

スタブを利用した Web サービスの呼び出しは、以下の３つの手順で実現できます。

1. WSDL を指定してスタブクラスを生成します。
2. Webサービス を呼び出す実行クラスを作成します。
3. 上記の実行クラスを利用して Web サービスにアクセスします。

このチュートリアルでは、「*Webサービス・プロバイダ の作成*」で解説されている Web サービスが呼び出されるまでを解説します。

開発環境を用意する

Webサービス・プロバイダ の「*開発環境を用意する*」と同様に開発環境（e Builder、Resin、および、intra-mart Accel Platform）をインストールします。
このチュートリアルでは、以下のプロジェクトを作成し、開発を行う手順を説明します。

グループID	mypackage （デフォルトの設定を使用）
バージョン	1.0.0 （デフォルトの設定を使用）
プロジェクト名	sample_client

e Builder のインストールが完了したら、以下の手順に従って、「Module Project」でプロジェクトを作成します。
詳しくは「intra-mart e Builder for Accel Platform アプリケーション開発ガイド」-「モジュール・プロジェクト作成」を参照してください。

1. 「ファイル」ー「新規」ー「プロジェクト」をクリックします。
2. 「e Builder」ー「Module Project」を選択して「次へ」をクリックします。
3. プロジェクト名に「sample_client」を入力して、「終了」をクリックします。

プロジェクトの作成が完了したら、intra-mart Accel Platform の API を使用できるようにするために、プロジェクトの設定を行います。
詳しくは「intra-mart e Builder for Accel Platform アプリケーション開発ガイド」-「プロジェクトの設定」を参照してください。

1. プロジェクトを右クリックして「プロパティ」を選択します。
2. 「e Builder」ー「Module Assembly」を選択します。
3. Web アーカイブディレクトリに、war を展開してできたコンテキストパスと同名のフォルダを選択します。
4. リソース変更時の自動デプロイ先の一覧で、全てのチェックボックスを外します。
5. 「OK」をクリックします。

依存関係を解決する

プロジェクトの設定が完了したら、依存関係の修正を行います。
Webサービス・プロバイダ にアクセスするためには、「Webサービス 認証・認可クライアント」モジュールに依存する必要があります。

以下の手順に従って、プロジェクトの依存関係を修正します。

1. 作成したプロジェクトのルートディレクトリに配置されている <module.xml> をダブルクリックします。
2. 「依存関係」タブを開き、「追加」をクリックします。
3. 以下の内容を入力して、「OK」をクリックします。

ID	jp.co.intra_mart.im_ws_auth_client
バージョン	8.0.2

> **コラム**
>
> 基本的にバージョンはサポートが行われている番号を指定します。
> 使用したい API が他のバージョンに含まれている場合、そのバージョン番号を指定してください。

4. <module.xml> ファイルを保存した後、「module.xml」タブを開き、不要なタグ（`<tags>`）を除去します。
最終的なソースは以下です。

```
<?xml version="1.0" encoding="UTF-8"?>
<module conf:schemaLocation="urn:intramart:jackling:toolkit:configurations configurations.xsd"
      xmlns="urn:intramart:jackling:module" xmlns:conf="urn:intramart:jackling:toolkit:configurations"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemeLocation="urn:intramart:jackling:module module.xsd">

   <id>mypackage.sample_client</id>
   <version>1.0.0</version>
   <type>module</type>
   <name>${module.name}</name>
   <vendor>${module.vendor}</vendor>
   <description>${module.description}</description>

   <dependencies>
      <dependency>
         <module-id>jp.co.intra_mart.im_ws_auth_client</module-id>
         <verified-version min="8.0.2">8.0.2</verified-version>
      </dependency>
   </dependencies>
</module>
```

5. 「依存関係の階層」タブを開き、依存関係が解決されさまざまなモジュールが表示されていれば成功です。

スタブクラスを作成する

スタブクラスを作成します。ここではWebサービス用ツールを利用して WSDL からスタブクラスを作成します。

この手順は以下の条件を満たしている環境で行うことを前提とします。

- Apache Ant がインストールされていること
- 「Webサービス用ツール」が存在すること

  「Webサービス用ツール」は プロダクトファイルダウンロード より <WebService_Tools.zip> をダウンロードしてください。
  ※ ダウンロードには製品のライセンスキーが必要です。
- Webサービス・プロバイダ が公開している WSDL が存在すること

Webサービス用ツールを展開したディレクトリを %WEBSERVICE_TOOL_HOME% として以降記述します。

> **コラム**
>
> ビルドツール「Ant」がインストールされていない場合は以下のサイトを参考にしてインストールを行ってください。
>
> Apache Ant Manual - Installing Apache Ant

1. スタブを生成するための各種設定を行います。

    1. StubGen.propertiesの編集を行います。
    %WEBSERVICE_TOOL_HOME%/StubGen.propertiesの編集を行います。
    `wsdlfilename` の値を `http://<HOST>:<PORT>/<CONTEXT_PATH>/services/SampleMemberInfoOperatorService?wsdl` に設定します。
    `<HOST>`、`<PORT>`、および、`<CONTEXT_PATH>` については、Webサービス・プロバイダ が起動しているアプリケーションサーバを指すものに変更してください。
    各プロパティの説明は以下の通りです。

プロパティ名	必須	説明
wsdlfilename	○	WSDL のファイルパス、またはURLを指定します。
destDir	○	スタブクラスを出力するディレクトリを指定します。
imartDir	○	デプロイされた intra-mart Accel Platform が展開されているディレクトリを指定します。 この設定はスタブを作成するために必要なライブラリをクラスパス上に展開するために行います。 例） /resin にResinが存在し、imart.war をデプロイし展開済みの場合、imartDir が指定するディレクトリは /resin/webapps/imart を指定します。

> **注意**
> `imartDir` に指定するディレクトリは Webサービス 認証・認可 モジュールがデプロイされている必要があります。

> **注意**
> Windows環境でパスを指定する場合は区切り文字を / または \ としてください。

    2. StubGenを実行するための環境情報を指定します。
    Apache Antがインストールされているディレクトリを指定します。
    %WEBSERVICE_TOOL_HOME%/StubGen.bat（UNIX系OSの場合はStubGeb.sh）を編集します。
    環境変数「ANT_HOME」に対してAntがインストールされているディレクトリを指定します。
    （Windows系OSの場合）

```
REM StubGen.bat
set ANT_HOME=C:/apache-ant
```

    （UNIX系OSの場合）

```
# StubGen.sh
export ANT_HOME=/apache-ant
```

2. StubGenを実行します。
    同梱されているバッチファイルを実行します。

    （Windows系OSの場合）

```
> %WEBSERVICE_TOOL_HOME%\StubGen.bat
```

    （UNIX系OSの場合）

```
$ sh %WEBSERVICE_TOOL_HOME%/StubGen.sh
```

StubGen.propertiesの destDir で指定されているディレクトリの配下にスタブクラス情報が出力されます。

> **コラム**
> StubGenは wsdlfilename で指定した WSDL が取得可能な状態で実行してください。

> **コラム**
> StubGenを実行すると、destDir に指定したディレクトリ配下に以下のスタブクラス情報が出力されます。
>
> - stub.jar：スタブのclassファイルが同梱されたjarファイル
> - classes ディレクトリ配下：スタブのclassファイル
> - src ディレクトリ配下：スタブのJavaファイル
>
> 必要に応じてそれぞれご利用ください。

Webサービス・クライアント を実装する

Webサービス・クライアント を実装します。

「*スタブクラスを作成する*」 で作成したスタブクラス情報をクラスパスに追加します。

1. e Builder プロジェクトを右クリックして、「ビルド・パス」ー「ビルド・パスの構成」を選択します。
2. 「*スタブクラスを作成する*」 で destDir 配下に出力された stub.jar を追加します。

以下の手順に従って、e Builder で Webサービス 実行クラスを作成します。

1. プロジェクトを右クリックして、「新規」ー「クラス」を選択します。
2. 以下の内容を入力して、「OK」をクリックします。

パッケージ	sample.web_service.client
名前	MemberInfoOperatorRunner

3. MemberInfoOperatorRunner.java がプロジェクトの src/main/java 配下に作成されます。
4. 以下を実装します。

```
package sample.web_service.client;

import java.rmi.RemoteException;
import java.text.SimpleDateFormat;
import java.util.Calendar;

import jp.co.intra_mart.foundation.web_service.util.impl.WSAuthDigestGenerator4WSSE;

import org.apache.axis2.AxisFault;

import sample.web_service.provider.SampleMemberInfoOperatorServiceStub;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.Add;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.Find;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.FindAll;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.FindAllResponse;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.FindResponse;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.Member;
import sample.web_service.provider.SampleMemberInfoOperatorServiceStub.WSUserInfo;

/**
 * スタブを利用してSampleMemberInfoOperatorServiceのオペレーションを実行するクライアントクラスのサンプルです。
 */
public class MemberInfoOperatorRunner {
```

```
// ホスト名、ポート番号、コンテキストパスは適宜置き換えてください。
private static final String ENDPOINT = "http://localhost:8080/imart/services/SampleMemberInfoOperatorService";

private static final String USER_CD = "aoyagi";

private static final String PASSWORD = "aoyagi";

// 実際にはプロバイダから提供された接続先ログイングループID/テナントIDを設定します。
private static final String LOGIN_GROUP_ID = "default";

public static void main(final String[] args) {
   try {
      new MemberInfoOperatorRunner().add();
   } catch (final RemoteException e) {
      e.printStackTrace();
   }
}

/**
 * SampleMemberInfoOperatorService#add を実行するサンプルです。<br>
 * 固定的にメンバー情報を追加します。
 */
public void add() throws RemoteException {
   // Webサービス・オペレーションへのパラメータを作成します。
   final Add parameter = new Add();
   parameter.setWsUserInfo(generateWSUserInfo());

   final Member member = new Member();
   member.setId("test");
   member.setName("テストユーザ");
   member.setAge(30);
   member.setMarried(true);
   final Calendar cal = Calendar.getInstance();
   cal.set(Calendar.YEAR, 1982);
   cal.set(Calendar.MONTH, Calendar.JUNE);
   cal.set(Calendar.DAY_OF_MONTH, 12);
   member.setBirthDate(cal);
   parameter.setMember(member);

   // Webサービスを実行します。
   final SampleMemberInfoOperatorServiceStub client = getStub();
   client.add(parameter);

   // 実行結果を標準出力します。
   System.out.println("Success.");
}

/**
 * SampleMemberInfoOperatorService#find を実行するサンプルです。<br>
 * ID "test" のメンバーを検索し、メンバー情報を標準出力します。
 */
public void find() throws RemoteException {
   // Webサービス・オペレーションへのパラメータを作成します。
   final Find parameter = new Find();
   parameter.setWsUserInfo(generateWSUserInfo());
   parameter.setId("test");

   // Webサービスを実行します。
   final SampleMemberInfoOperatorServiceStub client = getStub();
   final FindResponse response = client.find(parameter);
   final Member member = response.get_return();

   // 実行結果を標準出力します。
   printMember(member);
}

/**
 * SampleMemberInfoOperatorService#findAll を実行するサンプルです。<br>
 * 現在のメンバー情報の数を標準出力します。
 */
```

```
    public void findAll() throws RemoteException {
        // Webサービス・オペレーションへのパラメータを作成します。
        final FindAll parameter = new FindAll();
        parameter.setWsUserInfo(generateWSUserInfo());

        // Webサービスを実行します。
        final SampleMemberInfoOperatorServiceStub client = getStub();
        final FindAllResponse response = client.findAll(parameter);
        final Member[] members = response.get_return();

        // 実行結果を標準出力します。
        if (members == null) {
            System.out.println("Member count : 0");
        } else {
            System.out.println("Member count : " + members.length);
        }
    }

    private WSUserInfo generateWSUserInfo() {
        final WSUserInfo info = new WSUserInfo();
        info.setLoginGroupID(LOGIN_GROUP_ID);
        info.setUserID(USER_CD);
        info.setPassword(WSAuthDigestGenerator4WSSE.createWsseAuthString(USER_CD, PASSWORD));
        info.setAuthType(WSAuthDigestGenerator4WSSE.authType);

        return info;
    }

    private SampleMemberInfoOperatorServiceStub getStub() throws AxisFault {
        final SampleMemberInfoOperatorServiceStub client = new SampleMemberInfoOperatorServiceStub(ENDPOINT);

        return client;
    }

    private void printMember(final Member member) {
        System.out.println("id       : " + member.getId());
        System.out.println("name     : " + member.getName());
        System.out.println("age      : " + member.getAge());
        System.out.println("married  : " + member.getMarried());
        final SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
        System.out.println("birthday : " + formatter.format(member.getBirthDate().getTime()));
    }

}
```

コラム

Web サービスのパラメータとして、認証・認可用のユーザ情報を設定します。
このサンプルでは、認証タイプ「WSSE」を利用しています。
認証タイプ「WSSE」の詳細は「Webサービス 認証・認可 仕様書」-「認証・認可」を参照してください。

サンプルでは WSAuthDigestGenerator4WSSE クラスを利用してパスワード・ダイジェストを作成しています。
認証タイプ「WSSE」は、パスワードのダイジェスト化方法に、WS-Security の UsernameToken 形式を採用しています。
WSAuthDigestGenerator4WSSE クラスは、そのパスワード・ダイジェストの生成に特化したユーティリティです。
「ユーザコード」と「パスワード」を元にパスワード・ダイジェストを生成します。
WSAuthDigestGenerator4WSSEの詳細は、「WSAuthDigestGenerator4WSSEのAPIリスト」を参照してください。

コラム

wsUserInfoに設定する情報については、Webサービスプロバイダ側の設定と合わせる必要があります。
サンプルでは「generateWSUserInfo」メソッド内で、wsUserInfoの情報を定義しています。

-

コラム

Webサービス・プロバイダ がSOAPFaultエラーをスローした場合、スタブを実行した際に AxisFaultが例外としてスローされます。
AxisFaultの詳細は、「Axis2のAPIドキュメント」を参照してください。

なお、Webサービス・プロバイダ 側では、Web サービス呼び出し時に設定した認証情報（WSUserInfo）を元に認証・認可が行われます。
該当するユーザが存在しない、パスワードが間違っているなどの理由でユーザ情報が不正な場合、または、Web サービスを実行する権限がない場合には、AxisFaultが例外としてスローされます。
AxisFaultの `getFaultCode` メソッドでは、発生した問題に対応するコードが取得できます。
コードについての詳細は「Webサービス 認証・認可 仕様書」-「認証・認可のSOAPフォルト・コード」を参照してください。

Webサービス にアクセスする

e Builder上からJavaアプリケーションとして Webサービス にアクセスします。

- 以下の手順に従って、メンバ情報を登録します。

  1. プロジェクトの `src/main/java` ー `sample.web_service.client` ー `MemberInfoOperatorRunner.java` を右クリックし、「実行」ー「Java アプリケーション」を選択します。
  2. コンソールに以下のよう表示されれば成功です。

     ```
     Success.
     ```

     コラム

     Unsupported major.minor version エラーが発生する場合、Java コンパイラーのバージョンと実行時に利用する jre のバージョンが一致していることを確認してください。

- 以下の手順に従って、メンバ情報を検索します。

  1. 「*Webサービス・クライアント を実装する*」で作成したjavaクラスの `main` メソッドで実行するメソッドを `add` から `find` に変更します。
  2. プロジェクトの `src/main/java` ー `sample.web_service.client` ー `MemberInfoOperatorRunner.java` を右クリックし、「実行」ー「Java アプリケーション」を選択します。
  3. コンソールに以下のよう表示されれば成功です。

     ```
     id       : test
     name     : テストユーザ
     age      : 30
     barried  : true
     birthday : 1982-06-12
     ```

- 以下の手順に従って、メンバ情報の件数を表示します。

  1. 「*Webサービス・クライアント を実装する*」で作成したjavaクラスの `main` メソッドで実行するメソッドを `find` から `findAll` に変更します。
  2. プロジェクトの `src/main/java` ー `sample.web_service.client` ー `MemberInfoOperatorRunner.java` を右クリックし、「実行」ー「Java アプリケーション」を選択します。
  3. コンソールに以下のよう表示されれば成功です。

     ```
     Member count : 1
     ```

     コラム

     「add」で追加した分のメンバ情報の件数が表示されます。

項目

トラブルシューティング

https で提供されている WSDL を利用する場合

以下のような、WSDL が https で提供されている場合のスタブによるWebサービスの実行を行うためには、接続先のサーバ証明書の取得・登録が必要です。

- https://<HOST>/<CONTEXT_PATH>/services/SampleWebService?wsdl

以下の手順に従って、サーバ証明書の取得・登録を行います。

1. 接続先のサーバ証明書を取得します。
   サーバ証明書の取得方法はいくつかありますが、ここでは Windows 環境の Internet Explorer 9 を利用して証明書を取得する方法を示します。
   1. Internet Explorer 9 を開き、WSDL の URL を入力してアクセスします。
   2. Alt キーを押下してメニューバーを開き、「ツール」ー「インターネット オプション」を選択します。
   3. 「コンテンツ」タブを開き、「証明書」をクリックします。
   4. 取得したい証明書を選択して「エクスポート」をクリックします。
   5. ウィザードを進めてサーバ証明書ファイルを保存します。
2. JDK に含まれる keytool を利用して、サーバ証明書をキーストアに追加します。
   例： サーバ証明書ファイルが C:\temp\server.crt に保存されており、別名「sample_alias」でキーストアエントリに追加する場合

   ```
   keytool -import -alias sample_alias -file C:\temp\server.crt
   ```

   > **コラム**
   >
   > 上記コマンドを実行すると、ユーザのホームディレクトリの「.keystore」ファイルに、キーストアが作成されます。
   > keytool の詳細は、以下「JDK ドキュメントの keytool - 鍵と証明書の管理ツール」を参照してください。
   >
   > http://docs.oracle.com/javase/jp/7/technotes/tools/windows/keytool.html （日本語）
   > http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html （英語）
3. アプリケーションサーバの JVM のシステムプロパティに「javax.net.ssl.trustStore」を追加します。
   例： Resin のインストール先が C:\resin で、ユーザ名が user_name の場合
   C:\resin\conf\resin.properties

```
jvm_args : -Djavax.net.ssl.trustStore="C:\Users\user_name\.keystore"
```

> **コラム**
>
> すでに jvm_args が存在する場合は、末尾に半角空白で 1 文字空けて追記してください。

また、WSDL の URL が https で始まっていたとしても、WSDL に記述されているエンドポイントが https でない場合は、スタブ を利用する際に、明示的にエンドポイントを指定してください。

```
// スタブのコンストラクタの第 1 引数にエンドポイントを指定します。
new SampleWebServiceStub("https://localhost/imart/services/SampleWebService");
```

「指定した要求に失敗しました」が発生する場合

本現象が発生した場合に考えられる原因は、以下の通りです。

- 指定した Web サービスを実行する権限がない可能性があります。

解決方法の詳細は「Webサービス 認証・認可 仕様書」-「認証・認可のSOAPフォルト・コード」の「wsse:RequestFailed」を参照してください。

「指定した RequestSecurityToken を理解できません」が発生する場合

本現象が発生した場合に考えられる原因は、以下の通りです。

- 認証タイプに対応する認証モジュールが存在しない可能性があります。

解決方法の詳細は「Webサービス 認証・認可 仕様書」-「認証・認可のSOAPフォルト・コード」の「wsse:BadRequest」を参照してください。

「要求が無効か、形式が間違っています」が発生する場合

本現象が発生した場合に考えられる原因は、以下の通りです。

- SOAP ボディにユーザ情報が存在していない可能性があります。
- ユーザ情報が格納されている要素名が「wsUserInfo」とされていない可能性があります。
- Web サービスとして公開する Java クラスのコンパイル方法が誤っている可能性があります。

WSDL の URL をブラウザで開き、Web サービスの関数定義内の引数名を確認してください。

■正

```
<xs:element name="add">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="wsUserInfo" type="ax22:WSUserInfo" nillable="true" minOccurs="0"/>
      <xs:element name="member" type="ax24:Member" nillable="true" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

■誤

```
<xs:element name="add">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="param0" type="ax22:WSUserInfo" nillable="true" minOccurs="0"/>
      <xs:element name="param1" type="ax24:Member" nillable="true" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

上記のように、引数名が「param0」「param1」のようになっている場合は、Java クラスのコンパイル方法が誤っています。

解決方法の詳細は「Webサービス 認証・認可 仕様書」-「認証・認可のSOAPフォルト・コード」の「wsse:InvalidRequest」を参照してください。

サンプルコード

Webサービス のタイムアウト時間を指定する

Webサービス 実行時のタイムアウト時間を指定するにはスタブに対してタイムアウト時間を設定します。

```
final SampleWebServiceStub client = new SampleWebServiceStub(ENDPOINT);
// タイムアウトを５秒（５０００ミリ秒）に指定
client._getServiceClient().getOptions().setTimeOutInMilliSeconds(5000);
```

バイナリファイルを送受信するサンプル

Web サービスとして公開する Webサービス・プロバイダ の実装クラスのメソッドの引数、および、返却値の型に「byte[]」を指定することで、バイナリファイルを送受信できます。
引き渡されたバイト配列は自動的に Base64 にエンコードされ SOAP メッセージとして送受信されます。
以下では、バイナリファイルを送受信するサンプルを動作させるまでの手順を説明します。

Webサービス・プロバイダ を作成する

「*Webサービス・プロバイダ の作成*」と同様に Webサービス・プロバイダ を実装します。
このサンプルでは、「*開発環境を用意する*」と「*依存関係を解決する*」は割愛します。

`sample.web_service.provider.PublicStorageAccessService.java` を用意します。
このクラスに定義されているメソッド「loadFile()」、および、「saveFile()」を Web サービスとして公開します。
PublicStorageAccessService.java の内容は以下の通りです。クラスの作成方法については「*Web サービス処理を実装する*」を参照してください。

```
package sample.web_service.provider;

import java.io.IOException;

import jp.co.intra_mart.foundation.service.client.file.PublicStorage;
import jp.co.intra_mart.foundation.web_service.auth.WSUserInfo;

import org.apache.axis2.AxisFault;

public class PublicStorageAccessService {

   public byte[] loadFile(final WSUserInfo wsUserInfo, final String path) throws AxisFault {
      final PublicStorage storege = new PublicStorage(path);
      try {
         return storege.load();
      } catch (final IOException e) {
         throw AxisFault.makeFault(e);
      }
   }

   public Boolean saveFile(final WSUserInfo wsUserInfo, final String path, final byte[] data) throws AxisFault {
      final PublicStorage storege = new PublicStorage(path);
      try {
         storege.save(data);
         return Boolean.TRUE;
      } catch (final IOException e) {
         throw AxisFault.makeFault(e);
      }
   }

}
```

PublicStorageAccessServiceを Webサービス として公開するための services.xml を作成します。
services.xmlは `src/main/webapp/WEB-INF/services/sample_public_storage/META-INF/services.xml` に配置します。
services.xml の内容は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceGroup>
   <service name="SamplePublicStorageAccessService">
      <parameter name="ServiceClass">sample.web_service.provider.PublicStorageAccessService</parameter>
      <module ref="im_ws_auth"/>
      <messageReceivers>
         <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
         <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
      </messageReceivers>

      <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>

      <operation name="loadFile">
         <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>
      </operation>

      <operation name="saveFile">
         <parameter name="authz-uri">service://intra-mart.jp/public-resources/welcome-to-intramart</parameter>
      </operation>

   </service>
</serviceGroup>
```

「*Webサービス をデプロイする*」と同様にデプロイを行います。

Webサービス・クライアント を作成する

「*Webサービス・クライアント の作成*」と同様に Webサービス・クライアント を実装します。

このサンプルでは、「*開発環境を用意する*」は割愛します。

SamplePublicStorageAccessService にアクセスするためのスタブを作成します。
「*スタブクラスを作成する*」の通りに、スタブクラスを作成します。
WSDL のURLは http://<HOST>:<PORT>/<CONTEXT_PATH>/services/SamplePublicStorageAccessService?wsdl です。
<HOST>、<PORT>、および、<CONTEXT_PATH> については、Webサービス・プロバイダ が起動しているアプリケーションサーバを指すものに変更してください。

「*Webサービス・クライアント を実装する*」の手順と同様に、上記で作成したスタブクラス情報をクラスパスに追加します。

Webサービス 実行クラス sample.web_service.client.PublicStorageAccessRunner.java を作成します。
PublicStorageAccessRunner.java の内容は以下の通りです。

```
package sample.web_service.client;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.rmi.RemoteException;

import javax.activation.DataHandler;

import jp.co.intra_mart.foundation.web_service.util.impl.WSAuthDigestGenerator4WSSE;

import org.apache.axis2.AxisFault;

import sample.web_service.provider.SamplePublicStorageAccessServiceStub;
import sample.web_service.provider.SamplePublicStorageAccessServiceStub.LoadFile;
import sample.web_service.provider.SamplePublicStorageAccessServiceStub.LoadFileResponse;
import sample.web_service.provider.SamplePublicStorageAccessServiceStub.SaveFile;
import sample.web_service.provider.SamplePublicStorageAccessServiceStub.WSUserInfo;

/**
 * スタブを利用してSamplePublicStorageAccessServiceのオペレーションを実行するクライアントクラスのサンプルです。
 */
public class PublicStorageAccessRunner {

    // ホスト名、ポート番号、コンテキストパスは適宜置き換えてください。
    private static final String ENDPOINT = "http://localhost:8080/imart/services/SamplePublicStorageAccessService";

    private static final String USER_CD = "aoyagi";

    private static final String PASSWORD = "aoyagi";

    private static final String LOGIN_GROUP_ID = "default";

    public static void main(final String[] args) {
        try {
            new PublicStorageAccessRunner().loadFile();
        } catch (final IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * SamplePublicStorageAccessService#loadFile を実行するサンプルです。<br>
     * パブリックストレージの"sample/test.data"をカレントディレクトリの"temp.data"にコピーします。
     */
    public void loadFile() throws IOException {
        // Webサービス・オペレーションへのパラメータを作成します。
        final LoadFile parameter = new LoadFile();
        parameter.setWsUserInfo(generateWSUserInfo());
        parameter.setPath("sample/test.data");

        // Webサービスを実行します。
        final SamplePublicStorageAccessServiceStub client = getStub();
        final LoadFileResponse response = client.loadFile(parameter);
```

```
        final DataHandler handler = response.get_return();

        // カレントディレクトリのファイル"temp.data"に内容を保存します。
        final File file = new File("temp.data");
        final FileOutputStream fos = new FileOutputStream(file);
        final InputStream in = handler.getInputStream();

        final byte[] buff = new byte[1024];
        try {
            while (true) {
                final int len = in.read(buff);
                if (len != -1) {
                    fos.write(buff, 0, len);
                } else {
                    break;
                }
            }
            fos.flush();
        } finally {
            fos.close();
            in.close();
        }

        // 実行結果を標準出力します。
        System.out.println("Out file : " + file.getAbsolutePath());
    }

    /**
     * SamplePublicStorageAccessService#saveFile を実行するサンプルです。<br>
     * カレントディレクトリの"temp.data"をパブリックストレージの"sample/test.data"にコピーします。
     */
    public void saveFile() throws RemoteException, MalformedURLException {
        // Webサービス・オペレーションへのパラメータを作成します。
        final SaveFile parameter = new SaveFile();
        parameter.setWsUserInfo(generateWSUserInfo());
        parameter.setPath("sample/test.data");
        final File file = new File("temp.data");
        parameter.setData(new DataHandler(file.toURI().toURL()));

        // Webサービスを実行します。
        final SamplePublicStorageAccessServiceStub client = getStub();
        client.saveFile(parameter);

        // 実行結果を標準出力します。
        System.out.println("Success.");
    }

    private WSUserInfo generateWSUserInfo() {
        final WSUserInfo info = new WSUserInfo();
        info.setLoginGroupID(LOGIN_GROUP_ID);
        info.setUserID(USER_CD);
        info.setPassword(WSAuthDigestGenerator4WSSE.createWsseAuthString(USER_CD, PASSWORD));
        info.setAuthType(WSAuthDigestGenerator4WSSE.authType);

        return info;
    }

    private SamplePublicStorageAccessServiceStub getStub() throws AxisFault {
        final SamplePublicStorageAccessServiceStub client = new SamplePublicStorageAccessServiceStub(ENDPOINT);

        return client;
    }

}
```

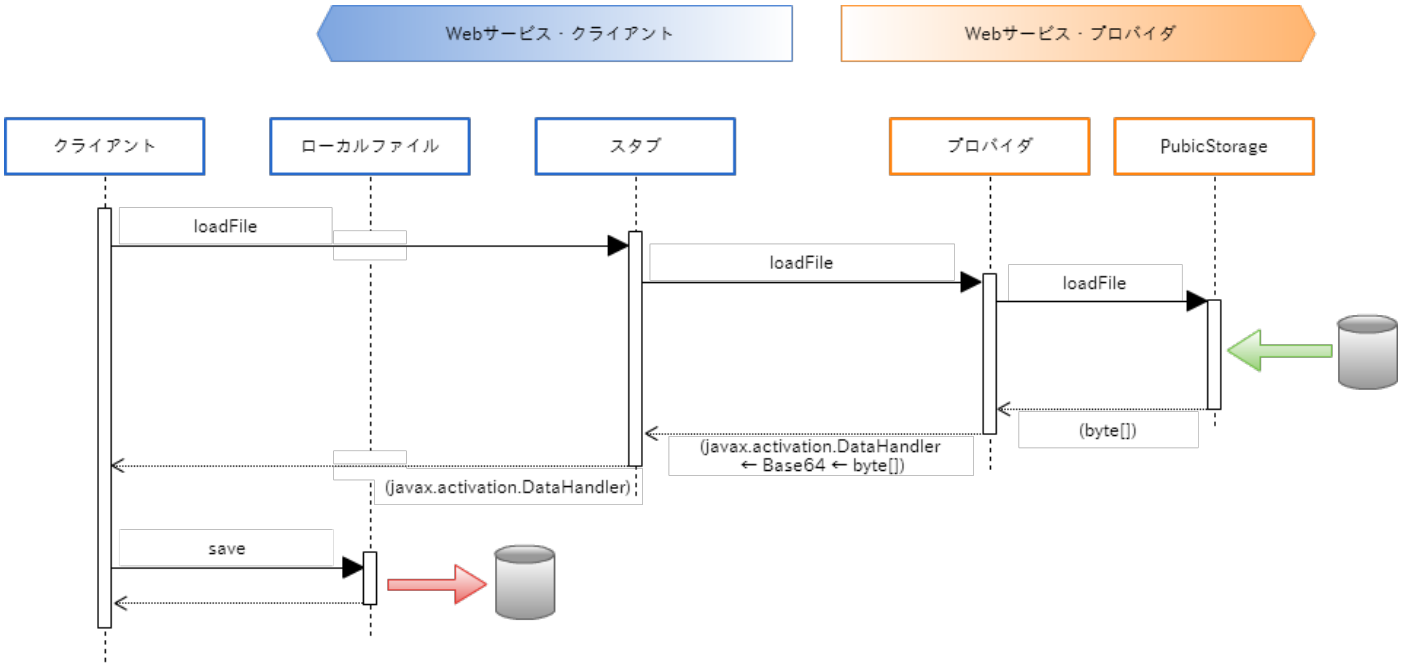
Webサービス を実行する

このサンプルプログラムは、Webサービス・クライアント のローカルファイルと Webサービス・プロバイダ 側の PublicStorage 内に

あるファイルの内容を送受信します。

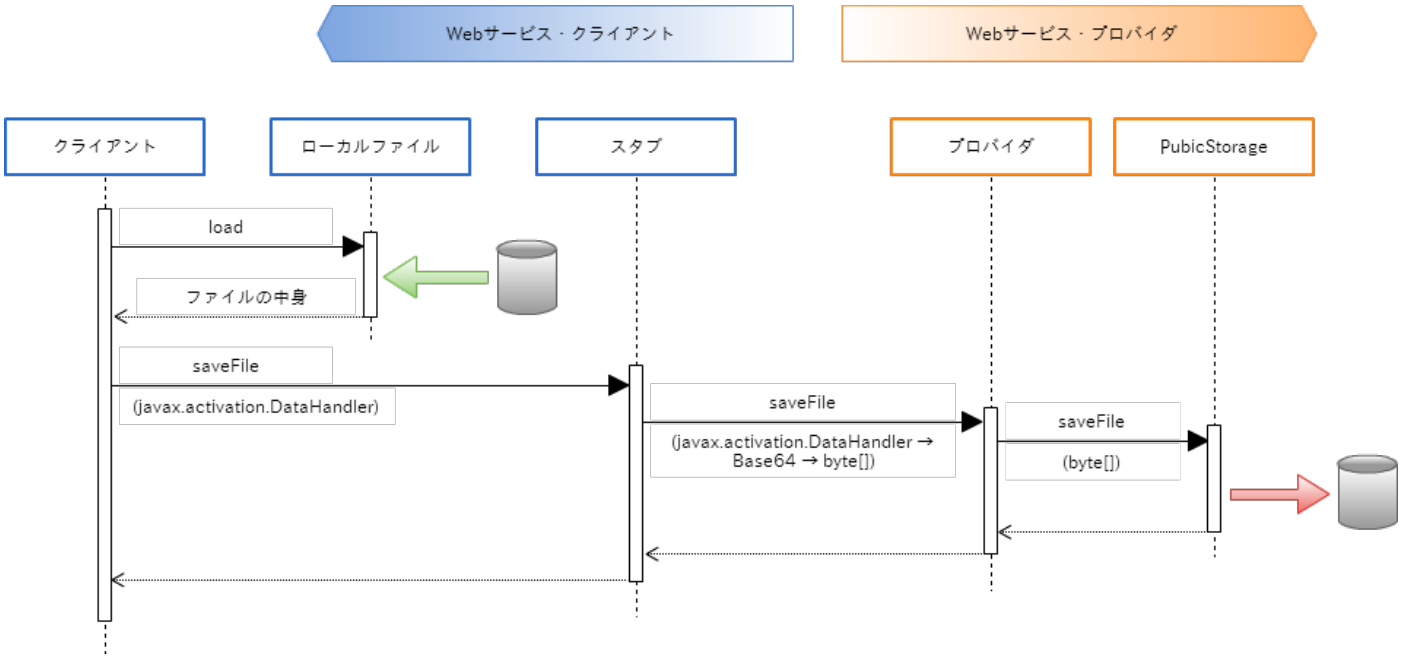
- バイナリファイルを受信する（SamplePublicStorageAccessService#loadFile）

  Web サービスを経由して Webサービス・プロバイダ 側の PublicStorage からファイルの中身を取得し、Webサービス・クライアント のローカルファイルに保存します。
  e Builder上で PublicStorageAccessRunner.java を右クリックし、「実行」－「Java アプリケーション」を選択します。
  loadFile を実行した場合の処理の流れは、以下の通りです。



- バイナリファイルを送信する（SamplePublicStorageAccessService#saveFile）

  Webサービス・クライアント のローカルファイルを読み取り、Web サービスを経由して Webサービス・プロバイダ 側の PublicStorage にファイルを保存します。
  PublicStorageAccessRunner.javaの main メソッドで実行するメソッドを loadFile から saveFile に変更します。
  e Builder上で PublicStorageAccessRunner.java を右クリックし、「実行」－「Java アプリケーション」を選択します。
  saveFile を実行した場合の処理の流れは、以下の通りです。



注意

Web サービスとして公開する関数の引数に JavaBean が指定されている場合、その JavaBean 内のバイト配列（byte[]）型のプロパティは、正常にデータが送受信されません。
これは Apache Axis2 の現行仕様による制限です。
バイナリファイルを送受信する場合は、JavaBean のプロパティではなく、Web サービスとして公開する関数の引数としてバイト配列（byte[]）を指定してください。