

intra-mart WebPlatform / AppFramework Ver.7.0

Web サービス・プログラミングガイド

2012/03/26 第8版

＜＜ 変更履歴 ＞＞

変更年月日	変更内容
2008/07/07	初版
2008/08/22	第 2 版 SOAPClient オブジェクトを、WebSphere および、WebLogic で利用する際の注意点を追記しました。 独自に作成した WSDL を WebSphere で利用する際の注意点を追記しました。 「7 制限事項 - 7.5 Webサービス・クライアント」の誤字を修正しました。
2008/09/30	第 3 版 「4.2.5 WSDLがhttpsで提供されている場合のSOAPClient利用方法」を追記しました。 「5.2.2.3.1 SSL経由のWebサービスを利用する場合」を追記しました。 「7 制限事項 - 7.5 Webサービス・クライアント」を追記しました。
2009/02/27	第 4 版 Axis2-1.4.1 に対応しました。
2009/06/30	第 5 版 「6.3.1 バイナリファイル送受信時の注意点」を追記しました。
2010/11/30	第 6 版 「3.2.1.1.1.1 WSDLにおけるユーザ情報の定義について」の誤字を修正しました。 「5.1.2.6.2.2 バッチファイルの実行」を修正しました。 「4.2.3 SOAPClientオブジェクトの設定」にスタブをコンパイルする際の最大ヒープサイズ設定を追加しました。 「7 制限事項 - 7.2 Axis2 - 1.4.xの現行仕様 - 16」を追記しました。
2011/06/30	第 7 版 「7.5 Webサービス・クライアント」にSOAPClientオブジェクトの返却値のプロパティ名に関する制限事項を追記しました。
2012/03/26	第 8 版 「4.2.3 SOAPClientオブジェクトの設定」の設定値「Never」の説明を修正しました。

<< 目次 >>

1	はじめに.....	1
1.1	目的.....	1
1.2	構成.....	1
2	Webサービス概要.....	2
2.1	Webサービスとは.....	2
2.1.1	Webサービス・プロバイダ と Webサービス・クライアント.....	2
2.2	SOAP.....	3
2.2.1	SOAPフォルト.....	3
2.3	WSDL.....	4
2.3.1	XMLスキーマ.....	4
2.3.2	Webサービス・オペレーション.....	4
2.4	Apache Axis2.....	5
2.4.1	Webサービス・プロバイダの簡単な例.....	5
3	Webサービスに対する認証・認可.....	6
3.1	機能概要.....	6
3.1.1	認証機能.....	6
3.1.2	認可機能.....	6
3.1.3	intra-martログインセッションの自動構築機能.....	6
3.1.4	用語解説.....	7
3.2	システム概要.....	8
3.2.1	認証・認可の流れ.....	8
3.3	認証モジュール.....	12
3.3.1	標準で用意されている認証モジュール.....	12
3.3.2	独自認証モジュールの利用方法.....	15
3.4	アクセス権限の設定.....	16
3.4.1	Webサービスアクセスの設定手順.....	16
3.5	認証・認可機能の設定.....	21
3.5.1	共通設定.....	22
3.5.2	各認証モジュールの設定.....	23
3.6	認証・認可機能のSOAPフォルト・コード一覧.....	24
3.6.1	wsse:InvalidRequest - 要求が無効か、形式が間違っています.....	24
3.6.2	wsse:BadRequest - 指定された RequestSecurityToken を理解できません.....	25
3.6.3	wsse:AuthenticationBadElements - ダイジェスト要素が不足しています.....	25
3.6.4	wsse:ExpiredData - 要求データが最新ではありません.....	25
3.6.5	wsse:InvalidSecurityToken - セキュリティ トークンが拒否されました.....	26
3.6.6	wsse: FailedAuthentication - 認証に失敗しました.....	26
3.6.7	wsse: RequestFailed - 指定した要求に失敗しました.....	27
4	チュートリアル(スクリプト開発モデル編).....	28
4.1	Webサービス・プロバイダの作成.....	28
4.1.1	概要.....	29
4.1.2	詳細手順.....	32
4.2	Webサービス・クライアントの作成.....	51
4.2.1	概要.....	51
4.2.2	詳細手順.....	51
4.2.3	SOAPClientオブジェクトの設定.....	58
4.2.4	WebLogicでSOAPClientオブジェクトを利用する方法.....	60

4.2.5	WSDLがhttpsで提供されている場合のSOAPClient利用方法	60
4.2.6	FAQ	62
5	チュートリアル (JavaEE開発モデル編)	65
5.1	Webサービス・プロバイダの作成	65
5.1.1	概要	66
5.1.2	詳細手順	67
5.2	Webサービス・クライアントの作成	78
5.2.1	概要	78
5.2.2	詳細手順	78
5.2.3	FAQ	82
6	開発時に有用な情報	84
6.1	発生しやすいエラーについて	84
6.1.1	「指定した要求に失敗しました」が発生します	84
6.1.2	「指定された RequestSecurityToken を理解できません」が発生します	84
6.1.3	「要求が無効か、形式が間違っています」が発生します	85
6.1.4	Webサービス・プロバイダでログインセッションが取得できません	85
6.2	SOAPメッセージのモニタリング	86
6.2.1	SOAPMonitor	86
6.2.2	TCPMon	87
6.3	バイナリファイルの送受信方法	89
6.3.1	バイナリファイル送受信時の注意点	89
6.4	Webサービスのデプロイ方法	90
6.4.1	services.xmlについて	90
6.4.2	ディレクトリ形式のデプロイ方法	91
6.4.3	aarファイル形式のデプロイ方法	91
6.5	Axis2 モジュールの適用方法	93
6.6	認証・認可を必要としないWebサービスの作成方法	93
6.7	SOAPフォルトの送信方法	94
6.8	Application Runtimeを分散させた場合のWSDLについて	95
7	制限事項	99
7.1	全般	99
7.2	Axis2 - 1.4.xの現行仕様	99
7.3	Webサービスに対する認証・認可	100
7.4	Webサービス・プロバイダ	101
7.5	Webサービス・クライアント	102

1 はじめに

1.1 目的

本書は、intra-mart における Web サービスの開発方法、および、intra-mart 上にデプロイされた Web サービスの認証・認可機能について説明します。

1.2 構成

第 2 章は、Web サービスの概要について述べています。

第 3 章は、intra-mart 上にデプロイされた Web サービスの認証・認可機能の詳細を記載しています。

第 4 章、および、第 5 章は、Web サービス・プロバイダ、および、クライアントの作成方法をチュートリアル形式で説明しています。

第 6 章では、開発時に有用な情報をまとめてあります。

第 7 章では、intra-mart で Web サービスを利用する際の制限事項が記載されています。

すぐに Web サービスを動作させてみたい方は、第 4 章、または、第 5 章からお読みください。

Web サービスをご存じない方は、まず第 2 章を読み、それから、第 4 章、または、第 5 章を読むことをお勧めします。

2 Web サービス概要

2.1 Webサービスとは

「Web サービス」という言葉は、コンピュータの世界の様々な場面で使われていますが、本書で扱う「Web サービス」とは、「SOAP/WSDL ベースの Web サービス」のことを意味します。簡単に言うと、XML 形式の情報を交換するためのルール「SOAP」と、Web サービスの仕様書である「WSDL (Web Services Description Language)」を用いた Web サービスのことを意味します。

2.1.1 Webサービス・プロバイダ と Webサービス・クライアント

本書では、Web サービスを提供する側を「Web サービス・プロバイダ」と呼び、Web サービスを利用する側を「Web サービス・クライアント」と呼びます。

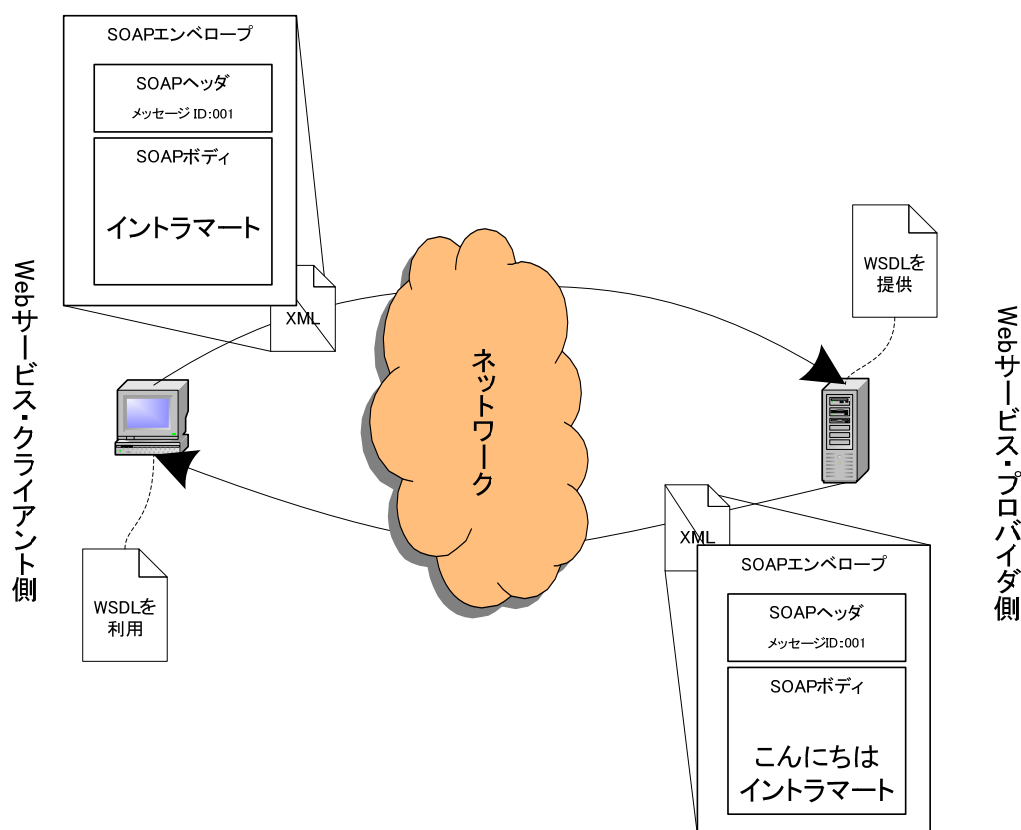


図 2-1 Web サービス概要

2.2 SOAP

SOAP とは、XML 形式のメッセージを交換するためのプロトコルです。Web サービスは、SOAP というルールに則って XML 形式のメッセージを交換しています。

SOAP では、XML 形式のメッセージを以下の二つの部分に分けています。

- SOAP ボディ
実際に交換されるメッセージの本文が記述される部分
- SOAP ヘッダ
メッセージ ID などの付加的な情報が記述される部分

そして、これら「SOAP ボディ」と「SOAP ヘッダ」の二つを「SOAP エンベロープ (= 封筒)」という概念で包んできます。 (「SOAP エンベロープ」のことを「SOAP メッセージ」と呼ぶ場合があります)

SOAP は、交換される XML メッセージのラッピング方法だけを規定したプロトコルです。SOAP エンベロープには、トランスポートプロトコルに関する記述はありません。SOAP は、メッセージ本文の種類やメッセージの配達方法に関わらず、封筒の形式だけを規定しているため、様々な場面で共通して利用することが可能です。

SOAPに関する詳細な情報は、書籍やWebサイトを参照してください。なお、SOAPの仕様は、W3Cの [Web Services Activity](http://www.w3.org/2002/ws/) (<http://www.w3.org/2002/ws/>) から入手可能です。

2.2.1 SOAPフォルト

SOAP では、Web サービス・プロバイダ側で何らかのエラーが発生した場合、SOAP ボディにエラー情報を格納して、Web サービス・クライアントに返信することができます。このエラー情報のことを「SOAP フォルト」と呼びます。SOAP フォルトには、エラーコードや、エラーメッセージ、および、エラー情報の詳細を格納することが可能です。

2.3 WSDL

WSDL (Web Services Description Language) とは、Web サービスを記述するための XML をベースとした言語仕様です。具体的には、Web サービスを利用する際に必要となる以下の情報が記述されています。

- Web サービスを利用するためにはどこにアクセスすればよいのか？
- Web サービスは、どんな通信プロトコルを使ってメッセージが交換されるのか？
- Web サービスを利用するために必要な入力情報は何か？
- Web サービスの実行結果はどのような形式で返却されるのか？

Web サービス・クライアントは、WSDL に記述された Web サービスのインタフェース情報を元に、Web サービスを利用します。

WSDL は、以下の要素から成り立っています。

- wsdl:definitions 要素
- wsdl:types 要素
- wsdl:message 要素
- wsdl:operation 要素
- wsdl:portType 要素
- wsdl:binding 要素
- wsdl:port 要素
- wsdl:service 要素

WSDLに関する詳細な情報は、書籍やWebサイトを参照してください。なお、WSDLの仕様は、W3Cの [Web Services Activity](http://www.w3.org/2002/ws/) (<http://www.w3.org/2002/ws/>) から入手可能です。

2.3.1 XMLスキーマ

WSDL の wsdl:types 要素は、Web サービスでやり取りされる XML メッセージのフォーマットを定義しています。ここで利用される XML メッセージの構造を定義するスキーマ言語として「XML Schema (=XML スキーマ)」が使われています。

XMLスキーマに関する詳細な情報は、書籍やWebサイトを参照してください。なお、XMLスキーマの仕様は、W3Cの [XML Schema](http://www.w3.org/XML/Schema) (<http://www.w3.org/XML/Schema>)から入手可能です。

2.3.2 Webサービス・オペレーション

Web サービス・オペレーションとは、Web サービス内で定義されている操作を意味します。Web サービス・オペレーションは、WSDL の wsdl: operation 要素で定義されています。

2.4 Apache Axis2

intra-martでは、Webサービスエンジンとして「[Apache Axis 2](http://ws.apache.org/axis2/index.html) (http://ws.apache.org/axis2/index.html)」を採用しています。Axis2を利用することにより、簡単にWebサービスを構築・利用することが可能です。

2.4.1 Webサービス・プロバイダの簡単な例

ここでは、ある文字列を送ると、その文字列に「こんにちは」という文字列を付け足して返却する簡単な Web サービスを作成します。(以降、intra-mart をインストールしたディレクトリを%IM_HOME%とします)

1. 「Hello.java」ファイルを作成します。

```
public class Hello {
    public String sayHello(String input) {
        return "こんにちは" + input;
    }
}
```

2. 上記をコンパイルし、%IM_HOME%/doc/imart/WEB-INF/classes/ディレクトリにコピーします。
3. %IM_HOME%/doc/imart/WEB-INF/services/hello_service/META-INF/ディレクトリを作成します。
4. %IM_HOME%/doc/imart/WEB-INF/services/hello_service/META-INF/services.xml ファイルを以下の内容で作成します。

```
<service name="HelloService" >
  <parameter name="ServiceClass">Hello</parameter>
  <messageReceivers>
    <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
      class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </messageReceivers>
</service>
```

以上で Web サービスを公開する準備が整いました。intra-martを起動し、Axis2 の管理コンソール画面 (http://host 名/imart/axis2-web/index.jsp) の[Services]リンクをクリックしてください。Web サービスとして公開されていることが確認できます。



3 Web サービスに対する認証・認可

3.1 機能概要

intra-mart では、Web サービスに対して認証、認可、および、intra-mart のログインセッションの自動構築を行うことが可能です。

3.1.1 認証機能

受信したユーザ情報を元に、Web サービス・クライアントが正しい intra-mart ユーザであると確認する機能です。Web サービスへのアクセスを、アカウントを持っているユーザのみに制限します。すなわち、認証に成功しなければ Web サービスを実行することは出来ません。これにより、ユーザベースのアクセス制御を行うことが可能になります。

3.1.2 認可機能

Web サービスのオペレーションに対して、アクセス権限を設定することが出来ます。これら権限と、認証されたユーザが持っているアクセス権限とを突き合わせ、利用可能な Web サービスを制限します。これにより、権限ベースのアクセス制御を行うことが可能になります。アクセス権限はロール単位で設定します。

3.1.3 intra-martログインセッションの自動構築機能

認証、認可が行われた後、intra-mart のログインセッションが構築されます。これにより、Web サービスの実装者は、ログイン処理を意識することなく、ビジネスロジックの公開を行うことが出来ます。加えて、intra-mart 上で構築された既存のビジネスロジックの再利用が容易になります。具体的には、DatabaseManager 等のログイングループに紐づく処理がそのまま Web サービスのビジネスロジックとして利用可能になります。

3.1.4 用語解説

3.1.4.1 認証とは？

認証とは、何らかの情報を元に、その対象の正当性を確認することです。本書では、送信されたユーザ情報を元に、Web サービス・クライアントが正しいユーザであると確認することを意味します。

3.1.4.2 認可とは？

認可とは、認証されたユーザが特定のリソースへのアクセスが出来るか否かを判定することです。

本書では、特定のリソースを「Web サービスのオペレーション」と位置づけます。

すなわち、あるユーザは、Web サービスのオペレーションを実行するための権限を持っている必要があります。

3.1.4.3 intra-martログインセッションとは？

intra-mart ログインセッションとは、大きく分けて二つの意味を持ちます。(つまり、intra-mart ログインセッションという言葉は、文脈によって意味が異なる場合があります)一つは、「intra-mart にログインした状態」を意味し、もう一つは、「intra-mart にログインしてからログアウトするまでの間に参照・操作が可能な値の保存領域」を意味します。

本書では、ある一般ユーザがログインしている状態のことを「intra-mart ログインセッションが構築されている」と表現したり、「intra-mart ログインセッションが存在する」と表現したりします。

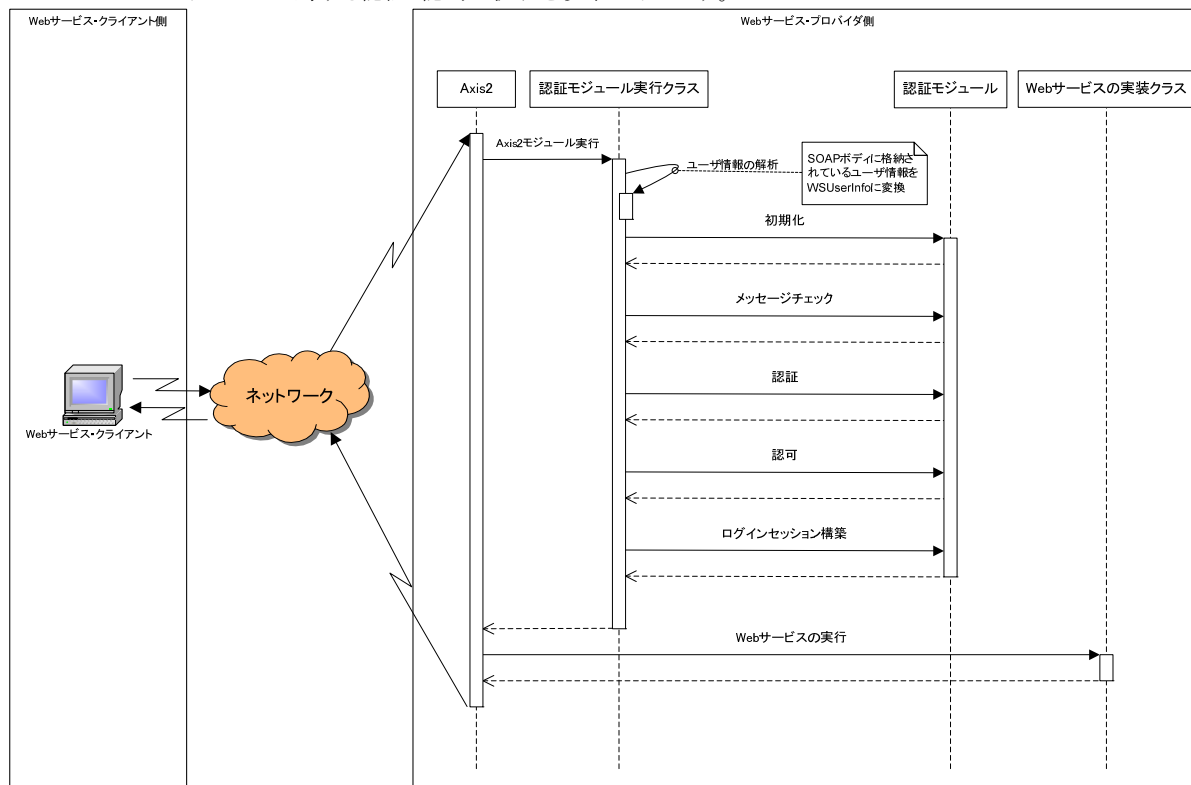
(その他に、ログイングループ ID を解決する際に「intra-mart ログインセッションからログイングループ ID を取得する」などと表現します)

なお、実装的な観点から言えば、intra-mart の API「AccessSecurityManager」を利用して、有効な intra-mart セッション情報を取得できる状態が、intra-mart にログインしている状態ということの意味します。

3.2 システム概要

3.2.1 認証・認可の流れ

Web サービスに対する認証・認可の流れを以下に示します。



intra-mart では、SOAP ボディに格納されたユーザ情報を元に認証・認可を行います。

ユーザ情報は、以下の4つで構成されています。

- 認証タイプ
- ログイングループ ID
- ユーザ ID
- パスワード

3.2.1.1 構成要素

3.2.1.1.1 ユーザ情報

intra-mart では、SOAP ボディに格納されたユーザ情報を元に認証・認可を行います。ユーザ情報は、「認証タイプ」、「ログイングループ ID」、「ユーザ ID」、「パスワード」の4つで構成されています。

ユーザ情報のパスワード(=WSUserInfoのpassword項目)に格納する内容は、認証タイプごとに異なります。詳しくは、「3.3 認証モジュール」を参照してください。

3.2.1.1.1.1 WSDL におけるユーザ情報の定義について

Web サービス・クライアントと Web サービス・プロバイダの双方で、ユーザ情報が SOAP ボディに格納されていることが理解できるように、上記で述べたユーザ情報を受け渡すためのメッセージ形式を WSDL に定義する必要があります。具体的には、下記の XML スキーマで定義される型で、名称が「wsUserInfo」という要素が、SOAP ボディの第 1 子要素の子要素として定義された WSDL を用意する必要があります。

```

・
・
<xs:schema attributeFormDefault="qualified"
    elementFormDefault="qualified"
    targetNamespace="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
  <xs:complexType name="WSUserInfo">
    <xs:sequence>
      <xs:element minOccurs="0" name="authType" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="loginGroupID" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="password" nillable="true" type="xs:string"/>
      <xs:element minOccurs="0" name="userID" nillable="true" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
・
・

```

Axis2 を利用することで、上記の定義を含んだ WSDL を自動生成することが可能です。

(WSDL を独自に用意する場合は、何もない状態から WSDL を作成するのではなく、Axis2 が自動生成した WSDL を再利用することが可能です)

Web サービス・オペレーションとして公開したいメソッドに、ユーザ情報を格納するための引数を追加します。
追加する引数は、型を「`jp.co.intra_mart.foundation.web_service.authentication.WSUserInfo`」クラス、名称を「`wsUserInfo`」として追加します。(大文字・小文字を厳密に判定します)

● Webサービスとして公開するクラス

```
import jp.co.intra_mart.foundation.web_service.auth.WSUserInfo;

public class Hello {
    public String sayHello(WSUserInfo wsUserInfo, String input){
        return "こんにちは + input;
    }
}
```

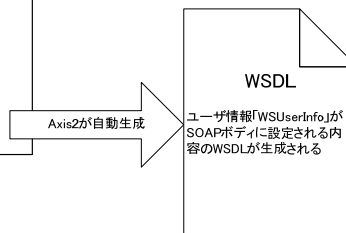
● ユーザ情報を格納するためのクラス

```
package jp.co.intra_mart.foundation.web_service.auth;

public class WSUserInfo implements Serializable {

    private String loginGroupID;
    private String userID;
    private String password;
    private String authType;

    (あとは、Setter/Getter)
    .
    .
    .
}
```



Web サービスの実装クラスを Axis2 で処理することにより、WSDL が自動生成されます。この結果、ユーザ情報が SOAP ボディに設定される内容の WSDL が生成されます。

3.2.1.1.2 認証モジュール実行クラス

認証モジュールを実行するための Axis2 ハンドラです。このハンドラは Web サービス・クライアントが送信したユーザ情報を解析後、認証タイプで特定される認証モジュールに認証・認可処理を委譲します。

なお、このクラスが動作するには、デプロイされているWebサービスに対して、Axis2 モジュール「`im_ws_auth`」が適用されている必要があります。Axis2 モジュールの適用方法に関しては「6.5 Axis2 モジュールの適用方法」を参照してください。

3.2.1.1.3 認証モジュール

intra-mart ユーザの認証、および、認可を行うためのモジュールです。

認証モジュールは、以下の順番で処理を行います。

1. 初期化处理
2. SOAP メッセージのチェック処理
3. 認証処理
4. 認可処理
5. ログイン処理

この一連の処理は、Webサービス・クライアントからの要求があるたびに実行されます。認証、および、認可などのすべての処理が成功した場合にのみ、指定されたWebサービスが実行されます。各処理のいずれかに失敗すると、エラーコードを格納したSOAPフォルトがクライアントに返却されます。エラーコードの詳細は、「3.6 認証・認可機能のSOAPフォルト・コード一覧」を参照してください。

3.2.1.1.4 Webサービスの実装クラス(Webサービス・プロバイダ)

Web サービスとして公開するビジネスロジックが記述されたクラスです。

認証モジュールの各処理がすべて正常終了した後に、クライアントが要求した Web サービス・オペレーションに対応するメソッドが実行されます。Web サービスの実装クラスは、認証、認可を考慮する必要なしに、intra-mart ログインセッションが構築された状態で実行されます。

intra-mart ログインセッションは、Web サービスの実行後、明示的に破棄(ログアウト)されます。したがって、intra-mart ログインセッションの範囲は Web サービスの呼び出し単位となります。

Web サービス・オペレーションの認証・認可を行うには、Web サービスとして公開するメソッドに対して、以下の二つの条件が満たされている必要があります。

- ユーザ情報を格納するための引数「wsUserInfo」が付与されていること
- Axis2 モジュール「im_ws_auth」が適用されていること

Axis2 モジュールの適用方法に関しては「6.5 Axis2 モジュールの適用方法」を参照してください。

3.2.1.1.5 Webサービス・クライアント

Webサービス・クライアント側では、「3.2.1.1.1.1 WSDLにおけるユーザ情報の定義について」で作られたWSDLからスタブを作成するなどして、Webサービス呼び出します。WSDLの中には、ユーザ情報を受け渡すためのメッセージ形式が定義されています。これに則って、Webサービス・クライアントはユーザ情報を設定します。

ユーザ情報の中には、認証タイプ(=WSUserInfo の「authType」項目)が用意されています。この項目によって、Web サービス・プロバイダ側で実行される認証モジュールが決定します。なお、認証タイプが未設定の場合、平文パスワード用認証モジュールが利用されます。(Web サービス・プロバイダ側で平文パスワード用認証モジュールが利用可能となっている必要があります)

送信されるパスワードの内容は、認証タイプごとに異なります。パスワードの内容を容易に生成するためのユーティリティ API が認証モジュールごとに提供されています。例えば、認証モジュール「WSSE」(詳細は後述)では、パスワードそのものではなく、ダイジェスト化されたパスワードを送信します。Web サービス・クライアントは、パスワード・ダイジェスト生成用のユーティリティ API 「WSAuthDigestGenerator4WSSE」を利用して、ユーザ情報の「password」にパスワード・ダイジェストを設定することになります。

3.3 認証モジュール

クライアントから送信される認証タイプによって、実行される認証モジュールが決定します。

ユーザ情報のパスワードに格納される文字列形式は認証モジュールごとに異なります。

認証モジュールは、「jp.co.intra_mart.foundation.web_service.auth.WSAuthModule」インタフェースを実装している必要があります。

認証タイプ	実装クラス名 ¹	概要
WSSE	WSAuthModule4WSSE	パスワードのダイジェスト化方法に、WS-Security の UsernameToken 形式を採用した認証モジュール。
BPMS	WSAuthModule4BPMS	BPMS 用の認証モジュール。
PlainTextPassword	WSAuthModule4PlainTextPassword	平文パスワード用認証モジュール。

3.3.1 標準で用意されている認証モジュール

3.3.1.1 認証タイプ「WSSE」

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 形式を採用した認証モジュールです。

実装クラスは「jp.co.intra_mart.foundation.web_service.auth.impl.WSAuthModule4WSSE」です。

Web サービス・クライアントは、クライアント自身が作成した「Nonce」「Created」および クライアントが管理している認証対象ユーザの「パスワード」を元にパスワード・ダイジェストを作成します。

WS-Security の UsernameToken 形式の認証用文字列(以降、WSSE 認証用文字列)の具体例を示します。(実際は一行です)

```
UsernameToken Username="the_who", PasswordDigest="tLDSsdGqfvraHRh8BpqTYRBVy+U=",  
Nonce="YTBiMWI20GI20TE3N2RI2Q==", Created="1966-12-01T12:34:56Z"
```

Web サービス・プロバイダ側の認証モジュールでは、WSSE 認証用文字列を解析し、クライアントから送られてきた「Nonce」「Created」および プロバイダ側で管理されている認証対象ユーザの「パスワード」を元にパスワード・ダイジェストを作成します。そして、「クライアントから送られてきたパスワード・ダイジェスト」と、「プロバイダ側で作成したパスワード・ダイジェスト」とを比較し、その結果が同一であれば該当ユーザであると判定し、そうでなければ、不正なユーザであると判定します。

¹ intra-mart が提供している認証モジュールの実装クラスは、全て jp.co.intra_mart.foundation.web_service.auth.impl パッケージに属しています。

WSSE 認証用文字列の各項目は以下の通りです。

項目	説明
Username	ユーザ名
Nonce	暗号的にランダムな値を Base64 エンコードした文字列
Created	Nonce が作成された日時を ISO-8601 表記で記述した文字列
PasswordDigest	「Nonce」「Created」 および 「パスワード」を文字列連結し、SHA1 アルゴリズムでダイジェスト化して生成された文字列を、Base64 エンコードした文字列 Password_Digest = Base64 (SHA-1 (nonce + created + password))

3.3.1.1.1 パスワード・ダイジェスト生成API「WSAuthDigestGenerator4WSSE」

認証タイプ「WSSE」に対応したパスワード・ダイジェスト生成用のユーティリティ API は以下の通りです。

詳細は API リストを参照してください。

開発モデル	API 名
スクリプト開発モデル	WSAuthDigestGenerator4WSSE オブジェクト
JavaEE 開発モデル	jp.co.intra_mart.foundation.web_service.util.impl.WSAuthDigestGenerator4WSSE (im_ws_auth_client.jar に含まれています)

jp.co.intra_mart.foundation.web_service.util.impl.WSAuthDigestGenerator4WSSE を利用する際には、クラスパスに以下の jar ファイルを追加してください。

- im_jdk_assist.jar
- im_ws_auth_client.jar

3.3.1.1.2 留意点

- Created を利用して有効期限チェックを行います。有効期限の初期値は 5 分です。システム日時と Created を比較し、有効期限が経過している場合、認証結果は常に NG となります。
 - 受信した WSSE 認証用文字列の Created と Nonce を有効期限までサーバ側で保持します。既に保持されている Created と Nonce で認証しようとした場合、その認証結果は常に NG となります。このチェックは、ApplicationRuntime 単位で行われます。
 - ユーザ情報(=WSAuthModule#authentication(WsUserInfo wsUserInfo)メソッドの引数)の「ユーザ ID」と「ログイングループ ID」を利用して認証対象ユーザを特定します。(WSSE 認証文字列の Username 項目は利用されません)
 - WSSE形式の認証用文字列の詳細は、「[Webサービスセキュリティ・ユーザネームトークン・プロファイル 1.0](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf) - 3 UsernameToken Extensions (http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf)」を参照してください。
- ☆ (上記の仕様 と この認証モジュールとの相違点は、「Nonce および Created 要素が必須であること」と「Nonce の符号化の種類が常に Base64 であること」です)

3.3.1.2 認証タイプ「BPMS」

認証タイプ「BPMS」は、BPMS 専用の認証モジュールです。

実装クラスは「jp.co.intra_mart.foundation.web_service.auth.impl.WSAuthModule4BPMS」です。パスワードのダイジェスト化方法やパスワード・ダイジェストの比較方法は認証タイプ「WSSE」と同一ですが、Username の形式が異なります。

認証タイプ「BPMS」利用時、ユーザ情報のパスワード(=WSUserInfo の password 項目)に格納する文字列(以降、BPMS 用 WS 認証ダイジェスト)の各項目は以下の通りです。

項目	説明
Username	「ログイングループ ID」+「区切り文字」+「ユーザ ID」 (例:default¥ueda) なお、区切り文字は設定で変更可能です。初期値は「¥」です。
Nonce	暗号的にランダムな値を Base64 エンコードした文字列
Created	Nonce が作成された日時を ISO-8601 表記で記述した文字列
PasswordDigest	「Nonce」「Created」および「パスワード」を文字列連結し、SHA1 アルゴリズムでダイジェスト化して生成された文字列を、Base64 エンコードした文字列 Password_Digest = Base64 (SHA-1 (nonce + created + password))

3.3.1.2.1 パスワード・ダイジェスト生成API「WSAuthDigestGenerator4BPMS」

認証タイプ「BPMS」に対応したパスワード・ダイジェスト生成用のユーティリティ API は以下の通りです。

詳細は API リストを参照してください。

開発モデル	API 名
スクリプト開発モデル	WSAuthDigestGenerator4BPMS オブジェクト
JavaEE 開発モデル	jp.co.intra_mart.foundation.web_service.util.impl.WSAuthDigestGenerator4BPMS (im_ws_auth_client.jar に含まれています)

jp.co.intra_mart.foundation.web_service.util.impl.WSAuthDigestGenerator4BPMS を利用する際には、クラスパスに以下の jar ファイルを追加してください。

- im_jdk_assist.jar
- im_ws_auth_client.jar

3.3.1.2.2 留意点

- Created を利用した SOAP メッセージの有効期限チェックは行われません。(ただし、設定により変更可能)
- Created と Nonce を利用したリプレイ攻撃対策は行われません。(ただし、設定により変更可能)
- BPMS 用 WS 認証ダイジェストの Username 項目に格納された「ユーザ ID」と「ログイングループ ID」を利用して認証対象ユーザを特定します。
(ユーザ情報(=WSAuthModule#authentication(WSUserInfo wsUserInfo)メソッドの引数)の「ユーザ ID」と「ログイングループ ID」は利用されません)

3.3.1.3 認証タイプ「PlainTextPassword」

認証タイプ「PlainTextPassword」は、パスワードを平文で送受信するための認証モジュールです。

3.3.1.3.1 留意点

- クライアントから送信される認証タイプ(=WSUserInfo の authType 項目)が未設定の場合、この認証モジュールが利用されます。
- この認証モジュールを利用するには、Webサービス・プロバイダ側で明示的に平文パスワード用認証モジュールが利用可能となっている必要があります。初期値は「利用不可」です。設定方法は、「3.5.1 共通設定」の「enablePlainTextPassword」を参照してください。
- この認証モジュールは、パスワードが平文で送信されるため、SOAP メッセージの中身を覗くことが出来る人物であれば、ユーザの成りすましが可能となることに注意してください。すなわち、この認証モジュールは、SSL のような外部のセキュアなシステムと併用されるのであれば、使用するべきではありません。

3.3.2 独自認証モジュールの利用方法

「jp.co.intra_mart.foundation.web_service.auth.WSAuthModule」インタフェースを実装したクラスを作成し、%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xml の<authModule>タグにそのクラスを設定することで、独自の認証モジュールを利用することも可能です。

%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xmlの<authModule>タグの詳細は、「3.5.1 共通設定」を参照してください。

本製品 CD-ROM の公開ソースファイル「im_ws_auth-src.zip」に標準で用意されている認証モジュールのソースコードが含まれています。あわせてご参照ください。

3.4 アクセス権限の設定

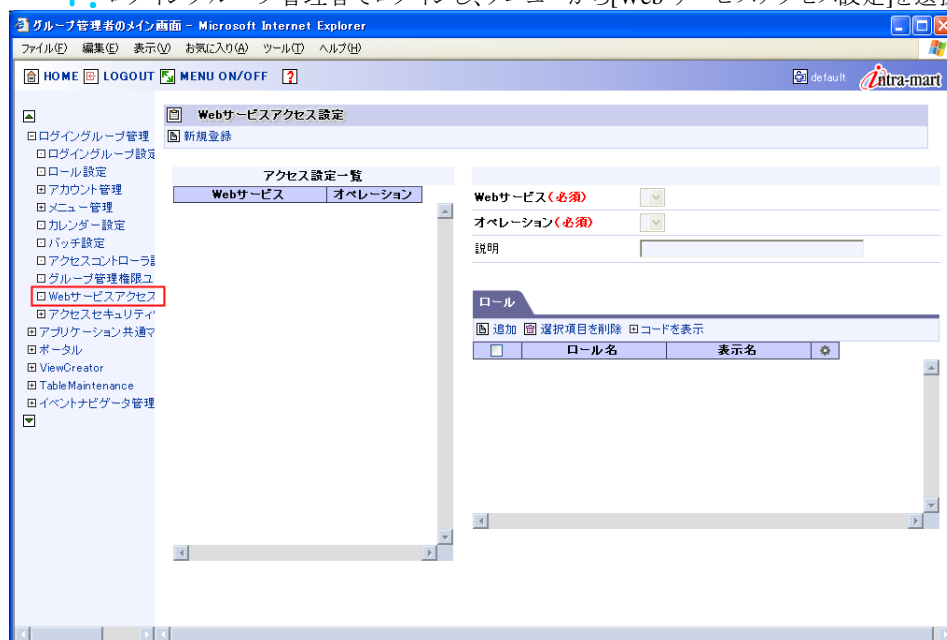
Web サービスのオペレーションに対してアクセス権限を設定するには、「Web サービスアクセス設定」メンテナンス画面を利用します。この画面では、登録されている任意の Web サービス・オペレーションの実行権限を、特定のロールに設定することができます。

Web サービスのアクセス権限情報はインポート、および、エクスポートすることが可能です。API「WSAccessManager」も用意されています。詳しくは「アクセスセキュリティ仕様書」を参照して下さい。

3.4.1 Webサービスアクセスの設定手順

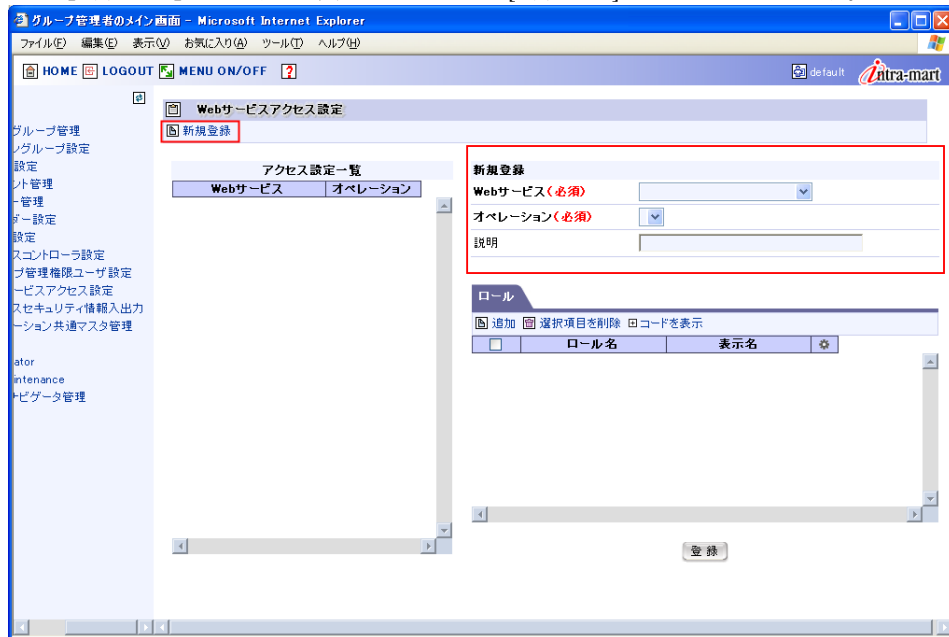
Web サービス・オペレーションの実行権限を新規登録する方法について説明します。

1. ログイングループ管理者でログインし、メニューから[Web サービスアクセス設定]を選択します。

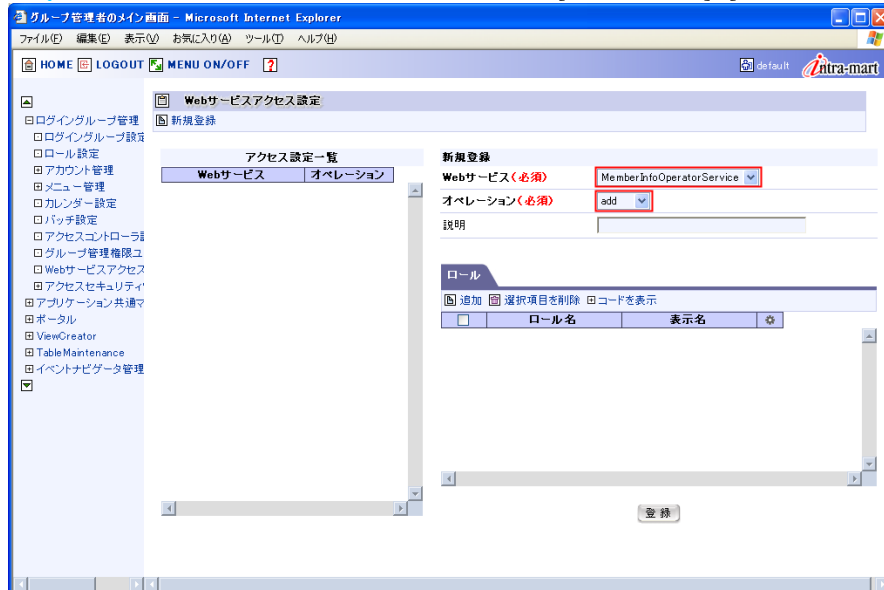


2. [新規登録]を選択してください。

[新規登録]を選択すると、右側のフレームに[新規登録]の画面が表示されます。



3. Web サービスとそのオペレーションをそれぞれ、[Web サービス]と[オペレーション]から選択します。



Web サービス(必須)
オペレーション(必須)
説明

登録されている Web サービスの一覧です。
上記の Web サービスが持つオペレーションです。
この Web サービスアクセス設定についての説明です。

4. 選択した Web サービス・オペレーションの実行権限を与えるロールの設定を行います。
新規で対象ロールを追加する場合は[ロール]タブ内の[追加]ボタンをクリックします。

ロール

追加

選択項目を削除

コードを表示

Web サービス・オペレーションを許可するロールの設定タブです。

Web サービス・オペレーションを許可するロールを追加することができます。

現在選択しているロールを削除することができます。

表示されているロール一覧にコードの列を表示させます。

5. [ロール検索]ウィンドウが表示されますので、任意のロールにチェックを入れ、[決定]ボタンを押下します。


カテゴリー	ロール名	表示名
<input type="checkbox"/>	guest	ゲストロール
<input checked="" type="checkbox"/>	level1	レベル1ユーザ
<input type="checkbox"/>	level2	レベル2ユーザ
<input type="checkbox"/>	level3	レベル3ユーザ
<input type="checkbox"/>	userSetting	ユーザ設定ロール

6. [ロール]タブに選択したロール一覧が表示されます。設定内容を確認し、[登録]ボタンを押下します。

ロール名	表示名
level1	レベル1ユーザ
level2	レベル2ユーザ
level3	レベル3ユーザ

※一覧からロールを削除するには、2 通りの方法があります。

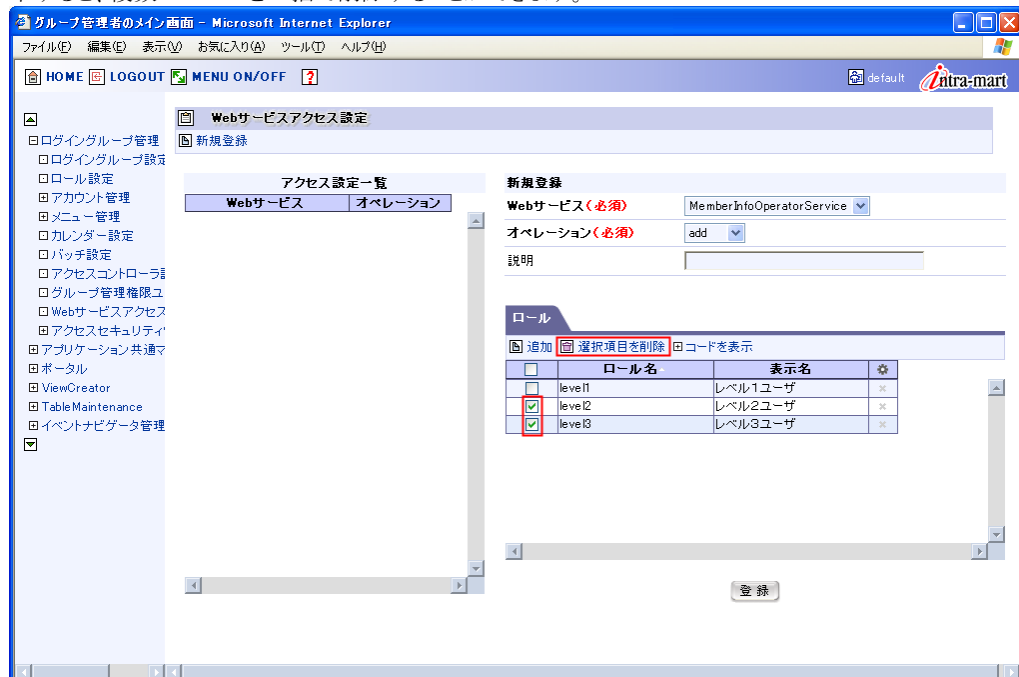
■ ワンクリック削除

右端のアイコン  をクリックすると、ロールが一行削除されます。

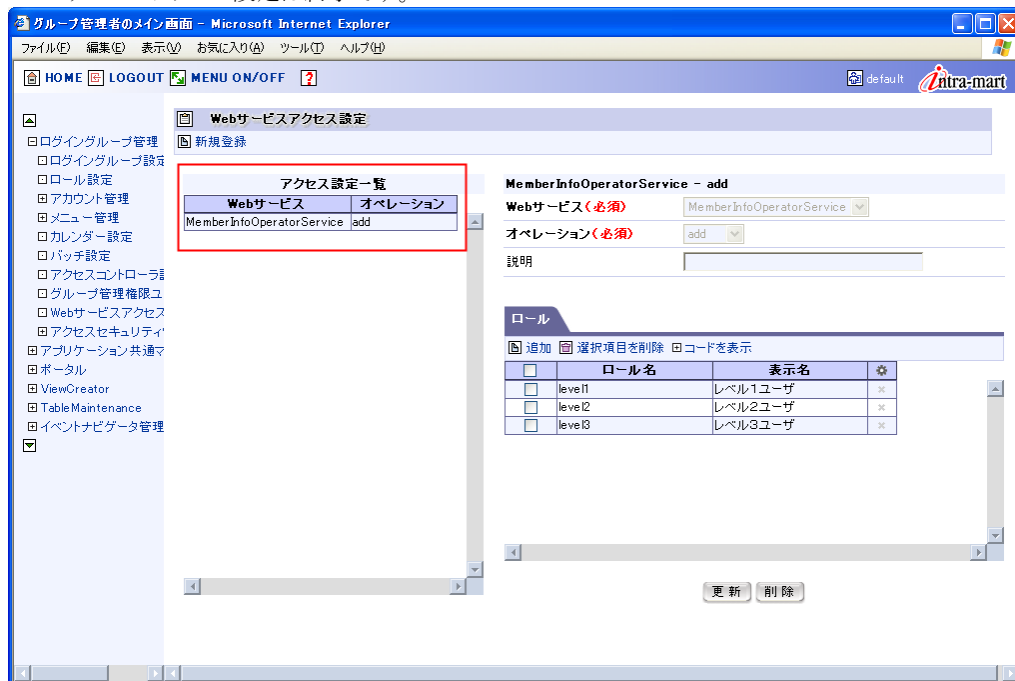


■ 一括削除

削除したいロールを選択し、テーブル左端のチェックボックスにチェックをつけます。[選択項目を削除]ボタンを押下すると、複数のロールを一括で削除することができます。



7. 画面左側の[アクセス設定一覧]に登録した Web サービスアクセス一覧が表示されます。これで、Web サービスアクセス設定は終了です。



3.5 認証・認可機能の設定

認証・認可機能の各種設定は、%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xml で行います。
設定は、parameter 名「jp.co.intra_mart.foundation.web_service」の子要素として記述します。

以下に、設定例を示します。

```
<axisconfig name="AxisJava2.0">

  <!-- ===== -->
  <!-- Parameters for intra-mart -->
  <!-- ===== -->
  <parameter name="jp.co.intra_mart.foundation.web_service">

    <enablePlainPassword>false</enablePlainPassword>

    <authModule class="jp.co.intra_mart.foundation.web_service.auth.impl.WSAuthModule4WSSE">
      <expire>300</expire>
    </authModule>

    <authModule class="jp.co.intra_mart.foundation.web_service.auth.impl.WSAuthModule4BPMS">
      <expire>0</expire>
      <usernameSeparator>#</usernameSeparator>
    </authModule>

    <enableAuthentication>true</enableAuthentication>
    <enableAuthorization>true</enableAuthorization>

    <showSoapFaultDetail>true</showSoapFaultDetail>
    <wsUserInfoArgumentName>wsUserInfo</wsUserInfoArgumentName>

  </parameter>
  .
  .
  .
```

3.5.1 共通設定

認証・認可機能の共通設定は以下の通りです。

設定項目	概要	初期値
authModule@class	<p>認証モジュールの実装クラス。</p> <p>ここで指定されるクラスは、 jp.co.intra_mart.foundation.web_service.auth.WSAuthModule インタフェースを実装する必要があります。</p> <p>WSAuthModule#setConfiguration(OMElement)メソッドには、 このタグ要素が引数に渡されます。</p>	なし
enablePlainTextPassword	<p>平文パスワード用認証モジュール利用可否設定。</p> <p>true の場合は利用可能、false の場合は利用不可となります。</p>	false
enableAuthentication	<p>認証機能(=ユーザベースのアクセス制御)の利用可否設定。</p> <p>true の場合は、ユーザベースのアクセス制御を行い、false の場合は、ユーザベースのアクセス制御は行いません。</p>	true
enableAuthorization	<p>認可機能(=権限ベースのアクセス制御)の利用可否設定。</p> <p>true の場合は、権限ベースのアクセス制御を行い、false の場合は、権限ベースのアクセス制御は行いません。</p>	true
showSoapFaultDetail	<p>認証モジュールで行われる認証、認可などの処理に失敗した際の例外スタックトレースを、SOAP フォルトの詳細要素に含めるか否かの設定。</p> <p>true の場合は、詳細情報に例外スタックトレースが付与され、false の場合は、付与されません。</p> <p>この設定は、Web サービスの実装クラスの実行時に発生した例外スタックトレースに関する設定ではありません。Web サービスの実装クラスから詳細情報を含んだ SOAP フォルトを返信する場合は、AxisFault 等を利用するなどして開発者が返信する必要があります。</p> <p>なお、スクリプト開発モデルでSOAPフォルトを返信する方法は、「6.7 SOAPフォルトの送信方法」で説明しています。</p>	true
wsUserInfoArgumentName	<p>ユーザ情報を格納するための引数名。</p> <p>この設定値は、認証モジュール実行クラスが SOAP ボディからユーザ情報を取得する際に利用されます。</p>	wsUserInfo

3.5.2 各認証モジュールの設定

各認証モジュール固有の設定は、authModule の子要素として設定します。例えば、認証タイプ「WSSE」で、ユーザ情報有効期限を設定する場合は、expire タグを authModule の子要素として記述します。

```
<axisconfig name="AxisJava2.0">
  <parameter name="jp.co.intra_mart.foundation.web_service">
    <authModule class="jp.co.intra_mart.foundation.web_service.auth.impl.WSAuthModule4WSSE">
      <expire>300</expire>
    </authModule>
  </parameter>
</axisconfig>
```

3.5.2.1 認証タイプ「WSSE」

設定項目	概要	初期値
expire	<p>認証時に利用されるユーザ情報の有効期限。単位は「秒」。</p> <p>システム日時と WSSE 認証文字列の Created を比較し、ここで設定された期限が過ぎている場合、認証結果が常に NG となります。</p> <p>また、WSSE 認証用文字列の Created と Nonce をここで設定された期間、サーバ側で保持します。既に保持されている Created と Nonce で認証しようとした場合、その認証結果は常に NG となります。(リプレイ攻撃対策)</p> <p>この設定が「0」の場合、有効期限チェック、および、Created と Nonce の保持は行われなくなります。</p>	300 (=5 分)

3.5.2.2 認証タイプ「BPMS」

設定項目	概要	初期値
expire	<p>認証時に利用されるユーザ情報の有効期限。単位は「秒」。</p> <p>システム日時と WSSE 認証文字列の Created を比較し、ここで設定された期限が過ぎている場合、認証結果が常に NG となります。</p> <p>また、WSSE 認証用文字列の Created と Nonce をここで設定された期間、サーバ側で保持します。既に保持されている Created と Nonce で認証しようとした場合、その認証結果は常に NG となります。(リプレイ攻撃対策)</p> <p>この設定が「0」の場合、有効期限チェック、および、Created と Nonce の保持は行われなくなります。</p>	0 (=有効期限チェックは行いません)
usernameSeparator	BPMS 用 WS 認証ダイジェストの「Username」の区切り文字。	¥ (=円サイン)

3.5.2.3 認証タイプ「PlainTextPassword」

この認証モジュール固有の設定はありません。

3.6 認証・認可機能のSOAPフォルト・コード一覧

認証・認可、および、intra-mart のログインセッション構築時にエラーが発生した場合、そのエラー内容に対応する SOAP フォルトがクライアントに返却されます。以下にその SOAP フォルト・コードと発生原因を記述します。

以降に記載されている「エラーメッセージ例」とは、エラー発生時に Web サービス・プロバイダ側で生成される例外のメッセージです。このメッセージは、例外スタックトレースを SOAP フォルトの詳細要素に含めるという設定が行われている場合、Web サービス・クライアントにも通知されます。（設定されていない場合は通知されません）

なお、この章に記述されていない SOAP フォルト・コードが送信された場合は、ユーザプログラム内で何らかのエラーが発生している可能性があります。つまり、認証・認可処理でのエラーは発生していないことを意味します。ユーザプログラム内で SOAP フォルト・コードが明示的に指定されていない場合、SOAP フォルト・コードは、名前空間「<http://www.w3.org/2003/05/soap-envelope>」で定義されている「Receiver」として送信されます。

ユーザプログラムで独自の SOAP フォルトを送信するには、AxisFault等を利用するなどして開発者が返信する必要があります。なお、スクリプト開発モデルで SOAP フォルトを返信する方法は、「6.7 SOAP フォルトの送信方法」で説明しています。

3.6.1 wsse:InvalidRequest - 要求が無効か、形式が間違っています

項目	説明
Fault Code	wsse:InvalidRequest
Fault Reason	要求が無効か、形式が間違っています。
エラーメッセージ例	java.lang.IllegalStateException: "wsUserInfo" is not found. java.lang.NullPointerException: SOAP body's first element is null.
発生原因	SOAP ボディにユーザ情報が存在しない、または、ユーザ情報が格納されている要素名が「wsUserInfo」ではない場合に発生します。 ユーザ情報が格納されている要素名が「param0」等の場合は、Web サービスとして公開する Java クラスが「-g」オプションなしでコンパイルされている可能性があります。 公開する Java クラスは、デバッグ情報を生成するようにコンパイルしてください。（javac の「-g」オプションを付与した状態でコンパイルを行ってください）これは、SOAP ボディに含まれるユーザ情報を「wsUserInfo」という名称で取得可能にするために必要な処理です。

3.6.2 wsse:BadRequest - 指定された RequestSecurityToken を理解できません

項目	説明
Fault Code	wsse:BadRequest
Fault Reason	指定された RequestSecurityToken を理解できません。
エラーメッセージ例	java.lang.IllegalStateException: WSAuthModule is not found. [authType="PlainTextPassword"]
発生原因	認証タイプに対応する認証モジュールが存在しない場合に発生します。 その他に、平文パスワード用認証モジュールを利用しない設定を行っている場合に、Web サービス・クライアントが認証タイプ未指定の状態で Web サービスの要求を行った場合にもこのエラーが発生します。

3.6.3 wsse:AuthenticationBadElements - ダイジェスト要素が不足しています

項目	説明
Fault Code	wsse:AuthenticationBadElements
Fault Reason	ダイジェスト要素が不足しています。
エラーメッセージ例	jp.co.intra_mart.foundation.web_service.auth.exception.AuthenticationBadElementsException: java.lang.IllegalStateException: No match found
発生原因	指定された認証タイプにおけるパスワード・ダイジェスト形式が間違っている場合に発生します。 例えば、認証タイプに「WSSE」が指定された場合、送信したパスワード・ダイジェストが WS-Security の UsernameToken 形式の認証用文字列として正しくない場合にこのエラーが発生します。(例:パスワード・ダイジェスト内に Nonce 項目が存在しない 等)

3.6.4 wsse:ExpiredData - 要求データが最新ではありません

項目	説明
Fault Code	wsse:ExpiredData
Fault Reason	要求データが最新ではありません。
エラーメッセージ例	jp.co.intra_mart.foundation.web_service.auth.exception.ExpiredDataException
発生原因	送信されたデータが設定された期限を過ぎている場合に発生します。 例えば、認証タイプ「WSSE」の場合、システム日時と WSSE 認証文字列の Created を比較し、設定された期限(初期値は 5 分)が過ぎている場合にこのエラーが発生します。

3.6.5 wsse:InvalidSecurityToken - セキュリティ トークンが拒否されました

項目	説明
Fault Code	wsse:InvalidSecurityToken
Fault Reason	セキュリティ トークンが拒否されました。
エラーメッセージ例	jp.co.intra_mart.foundation.web_service.auth.exception.InvalidSecurityTokenException
発生原因	送信されたデータが既に受信済みの場合に発生します。 例えば、認証タイプ「WSSE」の場合、WSSE 認証用文字列の Created と Nonce を設定された期間 (初期値 5 分) だけサーバ側で保持しています。既に保持されている Created と Nonce で認証しようとした場合、このエラーが発生します。

3.6.6 wsse: FailedAuthentication - 認証に失敗しました

項目	説明
Fault Code	wsse:FailedAuthentication
Fault Reason	認証に失敗しました。

このエラーはいくつかの発生原因があります。

3.6.6.1 エラーメッセージ「"ユーザID@ログイングループID" is not registered」

■ エラーメッセージ例

jp.co.intra_mart.foundation.web_service.auth.exception.AuthenticationException:
jp.co.intra_mart.foundation.security.exception.AccessSecurityException: "ued@default" is not registered.

■ 発生原因

ユーザ ID で特定されるユーザが存在しない場合に発生します。ログイングループ ID、および、ユーザ ID が正しいことを確認してください。

3.6.6.2 エラーメッセージ「"ユーザID@ログイングループID" has no license」

■ エラーメッセージ例

jp.co.intra_mart.foundation.web_service.auth.exception.AuthenticationException:
jp.co.intra_mart.foundation.security.exception.AccessSecurityException: "ued@default" has no license.

■ 発生原因

指定されたユーザのアカウントライセンスがありません。

3.6.6.3 エラーメッセージ「"ユーザID@ログイングループID" is expired」

- エラーメッセージ例

```
jp.co.intra_mart.foundation.web_service.auth.exception.AuthenticationException:
jp.co.intra_mart.foundation.security.exception.AccessSecurityException: "ued@default" is expired.
```

- 発生原因

指定されたユーザのアカウント有効期限が切れています。

3.6.6.4 エラーメッセージ「"ユーザID@ログイングループID" is locked」

- エラーメッセージ例

```
jp.co.intra_mart.foundation.web_service.auth.exception.AuthenticationException:
jp.co.intra_mart.foundation.security.exception.AccessSecurityException: "ued@default" is locked.
```

- 発生原因

指定されたユーザのアカウントがロックされています。

3.6.6.5 エラーメッセージ「Illegal PasswordDigest」

- エラーメッセージ例

```
jp.co.intra_mart.foundation.web_service.auth.exception.AuthenticationException:
Illegal PasswordDigest.
```

- 発生原因

指定されたパスワードが間違っています。認証タイプ「WSSE」では、「クライアントから送られてきたパスワード・ダイジェスト」と、「プロバイダ側で作成したパスワード・ダイジェスト」が同一でない場合に発生します。

3.6.7 wsse: RequestFailed - 指定した要求に失敗しました

項目	説明
Fault Code	wsse: RequestFailed
Fault Reason	指定した要求に失敗しました。
エラーメッセージ例	jp.co.intra_mart.foundation.web_service.auth.exception.AuthorizationException: <u>No access authority for web service operation:</u> SampleMemberInfoOperatorService#add()
発生原因	指定された Web サービス・オペレーションを実行する権限がない場合に発生します。 ログイングループ管理者の「Webサービスアクセス設定」メンテナンス画面を利用して、該当ユーザが保持するロールにWebサービス・オペレーションの実行権限を付与してください。設定方法は「3.4 アクセス権限の設定」を参照してください。

4 チュートリアル(スクリプト開発モデル編)

4.1 Webサービス・プロバイダの作成

この章では、スクリプト開発モデルのファンクションコンテナ(=サーバサイド JavaScript で記述されたビジネスロジック)を、Web サービスとして公開する手順を記します。

サーバサイド JavaScript で記述されたビジネスロジックを Web サービス化することで以下の利点が得られます。

- 既存のファンクションコンテナを Web サービスとして再利用可能
- 外部システムとの連携時に JavaScript が利用可能
- ロジック変更を即座に反映(再デプロイの必要なし)

この章を読み進める上での注意点は以下の通りです。

- Java クラスの作成に Eclipse を利用します。
スクリプト開発モデルおよび JavaEE 開発モデルのプログラム開発支援ツールである「intra-mart eBuilder」も利用可能です。
- JavaBean の簡単な知識が必要です。
 - ◇ JavaBean とは、再利用可能なプログラムを可能にするために、ある機能をまとめた Java のクラスです。JavaBean では、一定の作法に従って記述することにより、JavaScript のオブジェクトのようなプロパティという概念を実現しています。JavaBean では、プロパティの値を収めるフィールドを `private` で宣言し、そのフィールドの値を読み書きする「アクセッサ・メソッド(getter, setter メソッド)」を用意することでプロパティを定義します。

本書のスクリプト開発モデルによる Web サービス化に関しては、以下の型(および以下の型の配列)をプロパティとする JavaBean のみを扱います。

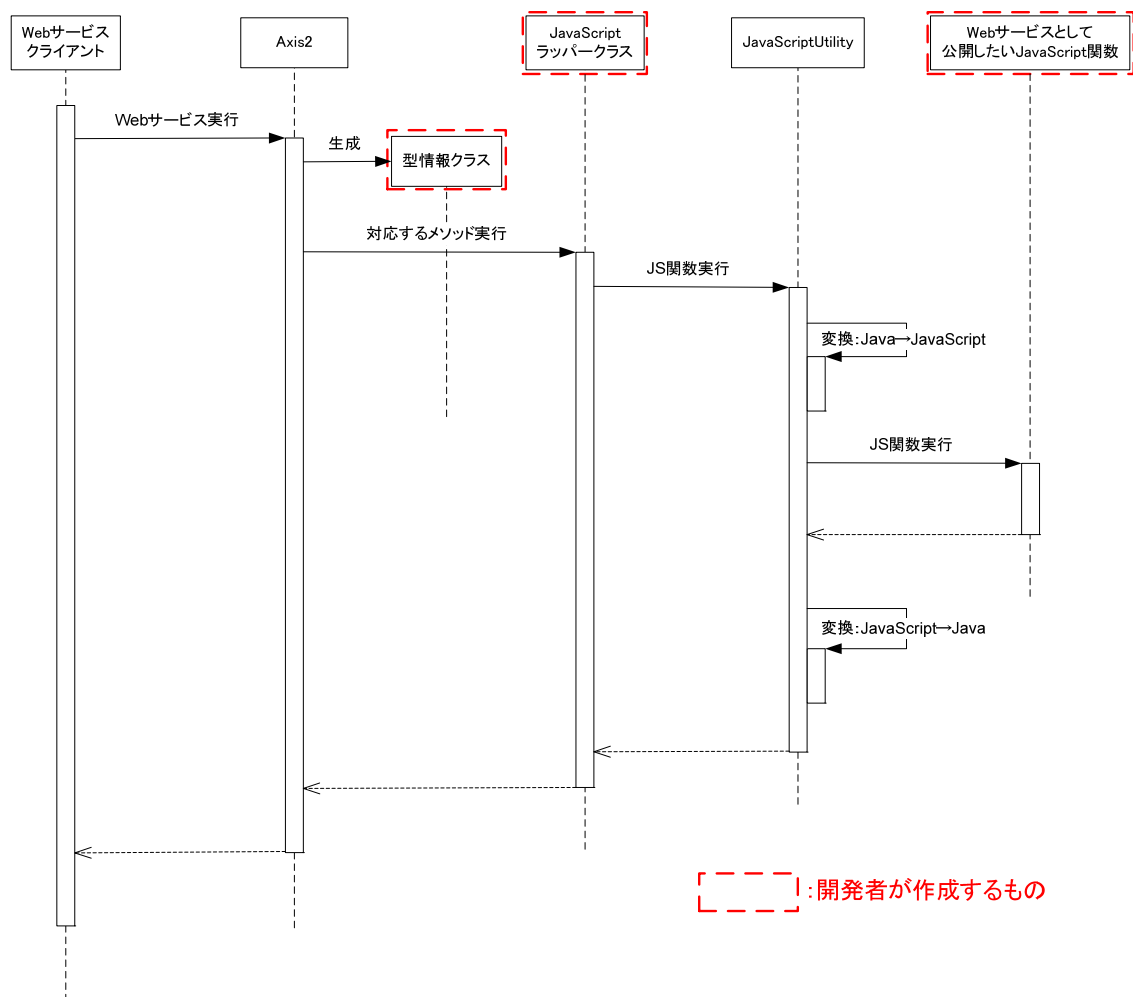
- `java.lang.String`
- `java.lang.Double`
- `java.lang.Boolean`
- `java.util.Date`
- `byte[]`

- ◇ 本書では、JavaBeanに関する「イベント」、および、「(getter, setter以外の)メソッド」は取り扱いません。JavaBeanの詳細は、[JavaBeans Component API](http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/beans/index.html) (<http://java.sun.com/j2se/1.5.0/ja/docs/ja/guide/beans/index.html>) を参照してください。

4.1.1 概要

JavaScript 関数の Web サービス化は、関数を実行するための Java クラスを作成し、その Java クラスを Web サービスとして公開することで実現します。(以降、この Java クラスを「**JavaScript ラッパークラス**」と呼びます)

Web サービスとして公開された JavaScript 関数が実行されるまでの流れを以下に示します。



- クライアントが、Web サービスの実行を要求します。
- Web サービス実行エンジン「Axis2」が、受け付けたリクエストに該当する JavaScript ラッパークラスのメソッドを呼び出します。
- ラッパークラスは「**JavaScriptUtility クラス**」を利用して JavaScript 関数を実行します。
- 以降、実行結果が適宜変換され、クライアントに返却されます。

開発者は、上記のシーケンス図の中で太い点線で囲まれた3種類のものを作成する必要があります。

一つ目は、「Web サービスとして公開したい JavaScript 関数」です。二つ目は、JavaScript 関数を実行するための「**JavaScript ラッパークラス**」です。三つ目は、JavaScript 関数の引数、および、返却値のオブジェクト変換に必要な「**型情報クラス**」です。「型情報クラス」とは JavaScript オブジェクトのプロパティ構成を、JavaBean で表現した単純なクラスです。

「JavaScriptUtility クラス」は、この「型情報クラス」を利用することで、Java 形式 から JavaScript 形式への変換、および、JavaScript 形式 から Java 形式へのオブジェクトを自動的に変換し、JavaScript 関数を実行します。

オブジェクト間のプロパティ変換規則は「4.1.1.1.1 プロパティ変換規則 : Java形式 → JavaScript形式」および「4.1.1.1.2 プロパティ変換規則 : JavaScript形式 → Java形式」を参照してください。

4.1.1.1.1 プロパティ変換規則 : Java形式 → JavaScript形式

Java		JavaScript
null	→	null
java.lang.String	→	String
java.lang.Number	→	Number
java.lang.Boolean	→	Boolean
java.util.Date	→	Date
java.util.Calendar		
byte[]	→	String (バイナリ)
javax.activation.DataHandler		
配列	→	Array

上記以外のクラスは変換対象外です。(例えば、List, Map, Set を変換することはできません)

JavaBean のプロパティの値が JavaBean 形式のインスタンスである場合、さらにそのプロパティの変換を試みます。
詳細は、API リスト「JavaScriptUtility」の javaBeanToJS(Object)メソッドの説明を参照してください。

4.1.1.1.2 プロパティ変換規則 : JavaScript形式 → Java形式

JavaScript		Java	備考
null	→	null	-
undefined			
String	→	java.lang.String	-
		java.lang.Character	先頭1文字を Character に変換します。
String (バイナリ)	→	byte[]	「String (バイナリ)」とは、JavaScriptAPI の「File オブジェクト」や「VirtualFile オブジェクト」の「load()関数」などで取得できるファイルデータ(バイナリ)の事を意味します。
		javax.activation.DataHandler	
Number	→	java.lang.Number	java.lang.Number のサブクラスが指定された場合は、その型に変換されます。変換可能な java.lang.Number のサブクラスは Double, Float, Long, Integer, Short, Byte の 6 クラスです。
Boolean	→	java.lang.Boolean	-
Date	→	java.util.Date	-
		java.util.Calendar	
Array	→	配列	JavaScript の Array 要素の型がすべて同じであり、かつ、その型が Java の配列要素の型に変換可能でなければなりません。
任意の JavaScript オブジェクト	→	java.lang.String	JavaScript オブジェクトの文字列表現が変換可能な場合に限る。
		java.lang.Number	
		java.lang.Boolean	

上記以外のクラスは変換対象外です。(例えば、List, Map, Set を変換することはできません)

JavaBean のプロパティの型が JavaBean 形式クラスの場合、さらにそのプロパティの変換を試みます。

詳細は、API リスト「JavaScriptUtility」の jsToJavaBean(Object, Class)メソッドの説明を参照してください。

4.1.2 詳細手順

この章では、スクリプト開発モデルのファンクションコンテナを Web サービスとして公開する手順を示します。

なお、この章で使用するサンプルはインストーラに収録されています。「サンプルをインストール」を選択し intra-mart をインストールしてください。Java のソースコードは、本製品 CD-ROM の公開ソースファイル「im_sample-src.zip」に含まれています。「6.3 バイナリファイルの送受信方法」のサンプルも収録されています。あわせてご参照ください。

4.1.2.1 Webサービス公開までの流れ

スクリプト開発モデルのファンクションコンテナを Web サービスとして公開するまでの流れは以下の通りです。

1. 準備
 - Web サービスとして公開する JavaScript 関数の選定
 - intra-mart のインストール
 - Eclipse のインストール
2. 型情報クラスの作成
 - JavaScript 関数のパラメータ 用
 - JavaScript 関数の返却値 用
3. JavaScript ラッパークラスの作成
4. Web サービスのデプロイ
 - jar ファイルの作成
 - aar ファイルの作成
5. アクセス権限の設定

4.1.2.2 準備

4.1.2.2.1 Webサービスとして公開するJavaScript関数の選定

Web サービスとして公開する JavaScript 関数を選定します。

ここでは、サンプルとして「sample/web_service/provider/member_info_operator.js」を用意します。

この JavaScript ソース内に定義されている関数「add()」、「find()」、および、「findAll()」を Web サービスとして公開します。

member_info_operator.js は、以下の形式のメンバー情報を操作します。

プロパティ	説明 (JavaScript 型)
id	メンバーID (String)
name	メンバー名 (String)
age	年齢 (Number)
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 (メンバー情報形式の配列)

member_info_operator.js のソースは以下の通りです。

```
var PREFIX = "SAMPLE_MEMBER_INFO_OPERATOR";

/**
 * メンバー情報の追加
 */
function add(member) {
    var message = "member_info_operator.js#add() が実行されました";
    Debug.console(message, member);

    // AccessSecurityManagerでログインユーザの情報が取得できます。
    var loginUserId = AccessSecurityManager.getSessionInfo().user;
    Debug.print("ユーザID「" + loginUserId + "」でログインしています");

    // 「.(ドット)」でプロパティにアクセスすることも可能です。
    Debug.print(member.name + "'s birthDate: " + member.birthDate);

    // Shared Memory Serviceに保存
    Module.external.set(PREFIX + member.id, member);
}

/**
 * メンバー情報の検索
 */
function find(id) {
    var message = "member_info_operator.js#find() が実行されました";
    Debug.console(message, id);

    // Shared Memory Serviceから読み込み
    var member = Module.external.get(PREFIX + id);
    return member;
}

/**
 * 全てのメンバー情報の検索
 */
function findAll() {
    var message = "member_info_operator.js#findAll() が実行されました";
    Debug.console(message);

    var memberArray = new Array();

    var allKeys = Module.external.keys();

    if(allKeys == null) {
        var soapFault = new SOAPFault("Shared Memory Serviceにデータが登録されていません");

        // SOAPFaultをスロー (ここで処理が終了します)
        soapFault.throwFault();
    }

    var max = allKeys.length;
    for(var idx = 0; idx < max; idx++) {
        var key = allKeys[idx];
        if(key.indexOf(PREFIX) == 0) {
            // Shared Memory Serviceから読み込み
            var member = Module.external.get(key);
            memberArray.push(member);
        }
    }
    return memberArray;
}
```

4.1.2.2.2 intra-martのインストール

intra-mart をインストールします。詳しくは、intra-mart のセットアップガイドを参照してください。

この章では、intra-mart WebPlatform (Resin)がスタンドアローンでインストールされている事とします。

以降、intra-mart がインストールされているディレクトリを **%IM_HOME%** とします。

4.1.2.2.2.1 重複するサンプルの削除

サンプルがインストールされている場合、以降で作成するサンプルコードが重複しますのでご注意ください。

重複を避けるためには、以下のファイル、および、ディレクトリを削除してください。

- %IM_HOME%/doc/imart/WEB-INF/services/im_ws_auth_sample/ ディレクトリ
- %IM_HOME%/doc/imart/WEB-INF/classes/sample/web_service/provider/Member.class
- %IM_HOME%/doc/imart/WEB-INF/classes/sample/web_service/provider/MemberInfoOperatorService.class

4.1.2.2.3 Eclipseのインストール

この章では、Java のクラスを作成するために Eclipse を利用します。Eclipse とは、オープンソースの統合ソフトウェア開発環境(IDE)です。なお、Eclipse ではなく、スクリプト開発モデルおよび JavaEE 開発モデルのプログラム開発支援ツールである「**intra-mart eBuilder**」を利用することも可能です。

[Eclipse.orgのダウンロードページ\(http://www.eclipse.org/downloads/index.php\)](http://www.eclipse.org/downloads/index.php) からEclipseをダウンロードします。

ここでは、「Eclipse IDE for java EE Developers」をダウンロードします。



ここでは、「eclipse-jee-europa-winter-win32.zip」をダウンロードした事とします。

ダウンロードしたアーカイブを解凍します。以降、この解凍先のパスを「**%ECLIPSE_HOME%**」とします。

日本語化が必要な場合は、Webサイト「[エクリプスWiki \(http://eclipsewiki.net/eclipse/index.php\)](http://eclipsewiki.net/eclipse/index.php)」内の [日本語化ブログ](#) ページを参考に日本語化を行うと良いでしょう。この章では、Pleiadesを利用して日本語化を行った Eclipseを利用します。

4.1.2.3 型情報クラスの作成

JavaScript 関数の引数、および、返却値のオブジェクト変換に必要な「型情報クラス」を作成します。
「型情報クラス」とは JavaScript オブジェクトのプロパティ構成を、JavaBean で表現した単純なクラスです。

ここでは、member_info_operator.js で利用するメンバー情報の形式を JavaBean として作成します。

4.1.2.3.1 継承関係を持つ型情報クラスの制限事項

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、継承関係を持ったクラスを指定することはできません。利用した場合、Web サービス・クライアント側でエラーとなる場合があります。Web サービス・クライアントとして、スクリプト開発モデル API「SOAPClient オブジェクト」を利用した場合、「ADBException: Unexpected subelement XXXX(=要素名)」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という、ADB (Axis Data Binding) の現行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、返却値として利用できません。

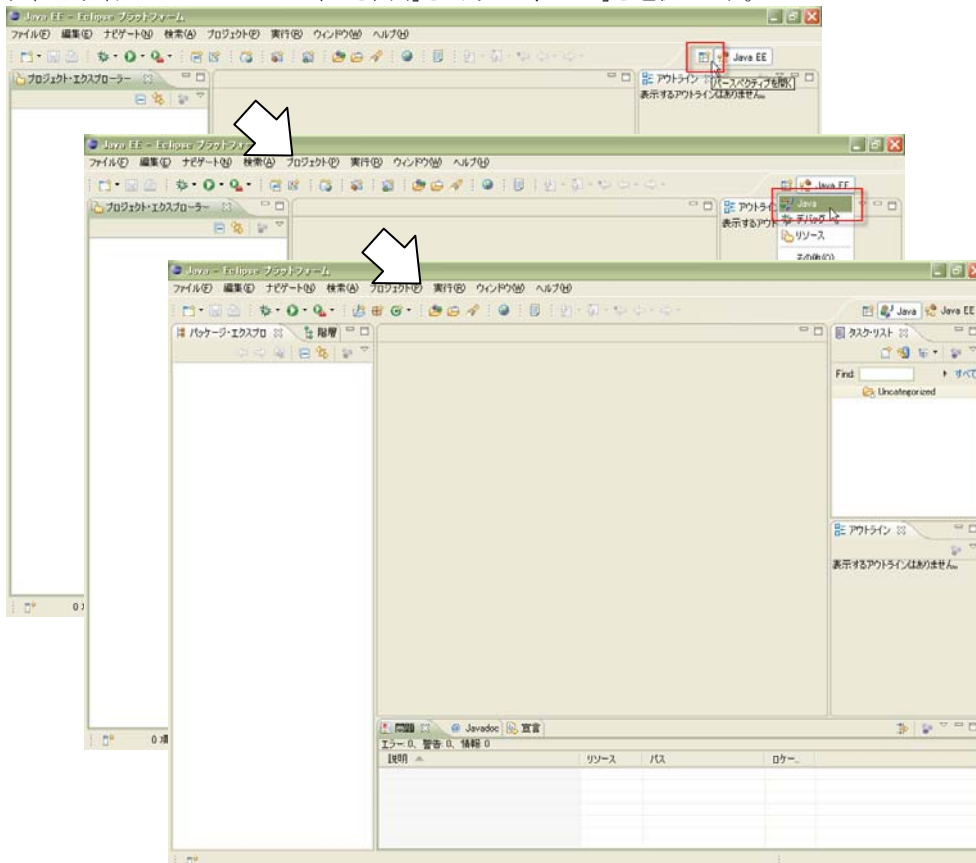
- sample.foo.ParantModel
- sample.bar.SubModel

4.1.2.3.2 Eclipseを起動し、Javaプロジェクトを作成

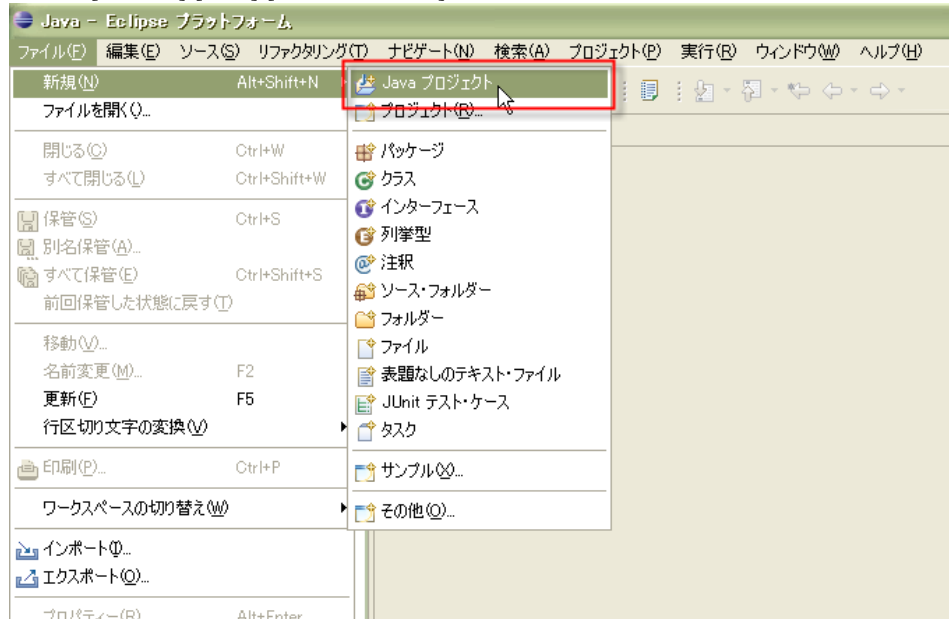
Eclipse を起動します。起動後、「ようこそ」が表示される場合はタブを閉じてください。



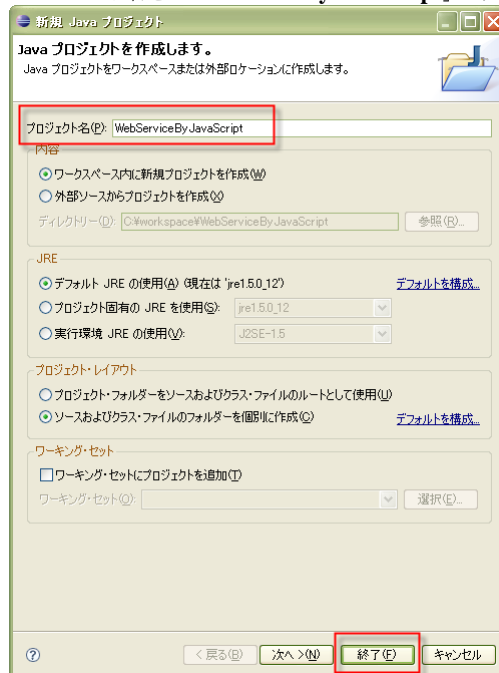
ウィンドウ右上の「パースペクティブを開く」をクリックし、「Java」を選択します。



メニュー[ファイル]-[新規]-[Java プロジェクト]を選択し、新規 Java プロジェクトを作成します。

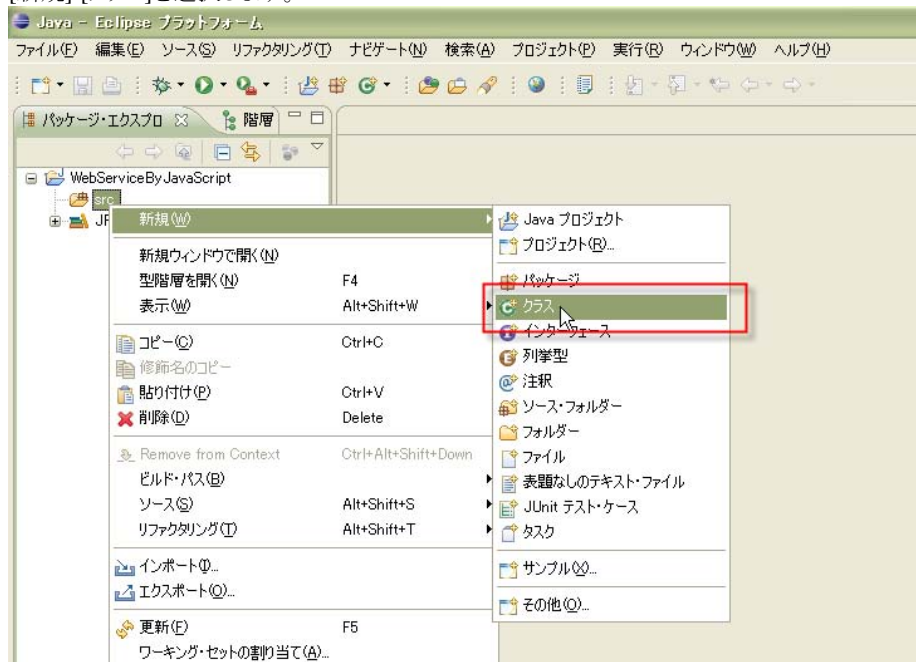


プロジェクト名を「WebServiceByJavaScript」と入力し、[終了]ボタンをクリックします。

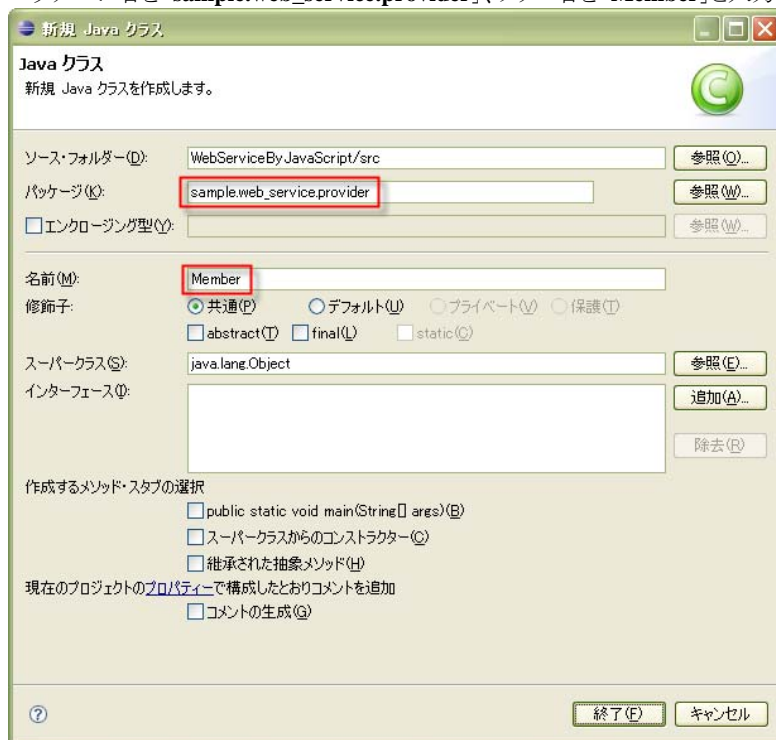


4.1.2.3.3 型情報クラスを新規作成

[パッケージ・エクスプローラ]ビューの「WebServiceByJavaScript」プロジェクト内の「src」ディレクトリ上で右クリックし、[新規]-[クラス]を選択します。



パッケージ名を「sample.web_service.provider」、クラス名を「Member」と入力して[終了]ボタンをクリックします。



Member.java が以下のように作成されます。

```
package sample.web_service.provider;  
  
public class Member {  
  
}
```

4.1.2.3.4 型情報クラスにプロパティを追加

メンバー情報のプロパティを追加します。

4.1.2.3.4.1 プライベート変数の定義

まず、プライベート変数を定義します。

具体的には、Member.java を以下のように編集します。(■が追加した箇所です)

```
package sample.web_service.provider;  
  
import java.util.Date;  
  
public class Member {  
    private String id;  
    private String name;  
    private Double age;  
    private Boolean married;  
    private Date birthDate;  
    private Member[] children; // Memberクラス形式の配列  
}
```

プロパティが JavaScript の Number 型の場合、Double クラスの変数として定義してください。
上記では、変数「age」が該当します。

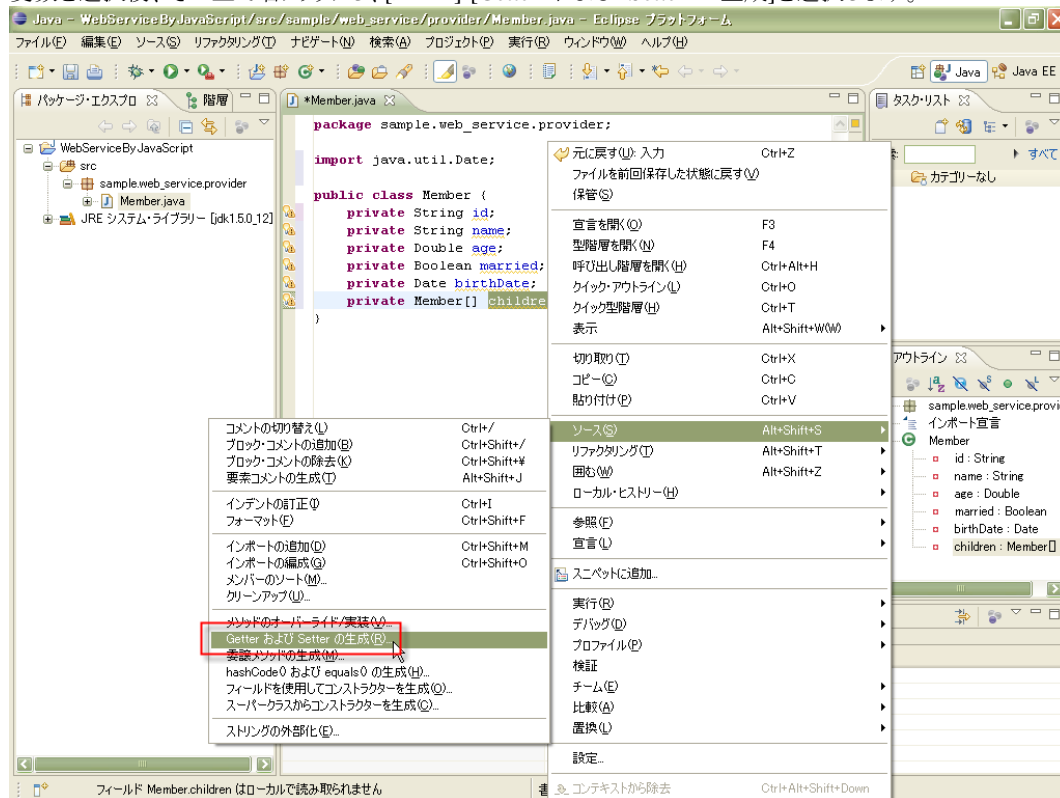
この Member クラスが、JavaScript 関数の引数、および、返却値のオブジェクト変換に必要な「型情報クラス」です。
「型情報クラス」とは JavaScript オブジェクトのプロパティ構成を、JavaBean で表現した単純なクラスです。

ここでは、member_info_operator.js で利用するメンバー情報の形式(「4.1.2.2.1 Web サービスとして公開する JavaScript 関数の選定」参照)を、JavaBean で表現した「Member クラス」として作成しています。

4.1.2.3.4.2 アクセッサ・メソッド(getter / setter)の追加

次にアクセッサ・メソッド(getter/setter)を追加します。

変数を選択後、その上で右クリックし、[ソース]-[Getter および Setter の生成]を選択します。



[すべて選択]ボタンをクリックし、変数すべてに getter/setter が作成されることを確認し[OK]ボタンをクリックします。



Member.java が以下のように変更されました。(が追加された箇所です)

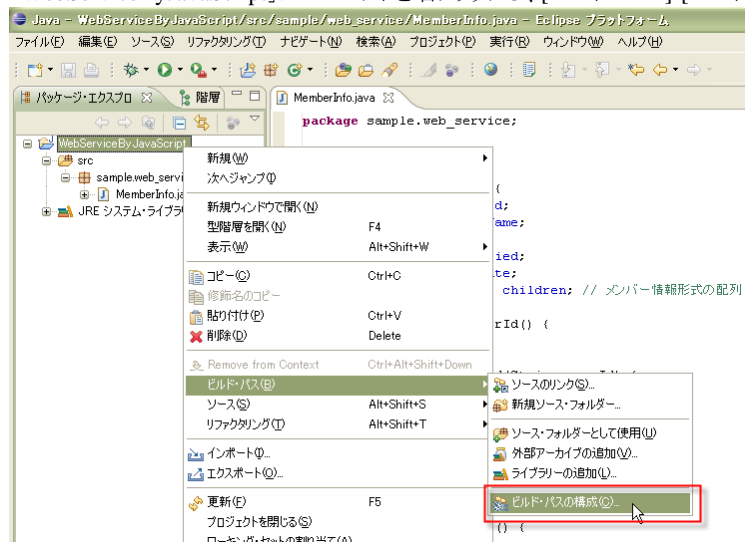
```
1: package sample.web_service.provider;
2:
3: import java.util.Date;
4:
5: public class Member {
6:     private String id;
7:     private String name;
8:     private Double age;
9:     private Boolean married;
10:    private Date birthDate;
11:    private Member[] children; // Memberクラス情報形式の配列
12:
13:    public String getId() {
14:        return id;
15:    }
16:    public void setId(String id) {
17:        this.id = id;
18:    }
19:    public String getName() {
20:        return name;
21:    }
22:    public void setName(String name) {
23:        this.name = name;
24:    }
25:    public Double getAge() {
26:        return age;
27:    }
28:    public void setAge(Double age) {
29:        this.age = age;
30:    }
31:    public Boolean getMarried() {
32:        return married;
33:    }
34:    public void setMarried(Boolean married) {
35:        this.married = married;
36:    }
37:    public Date getBirthDate() {
38:        return birthDate;
39:    }
40:    public void setBirthDate(Date birthDate) {
41:        this.birthDate = birthDate;
42:    }
43:    public Member[] getChildren() {
44:        return children;
45:    }
46:    public void setChildren(Member[] children) {
47:        this.children = children;
48:    }
49: }
```

4.1.2.4 JavaScriptラッパークラスの作成

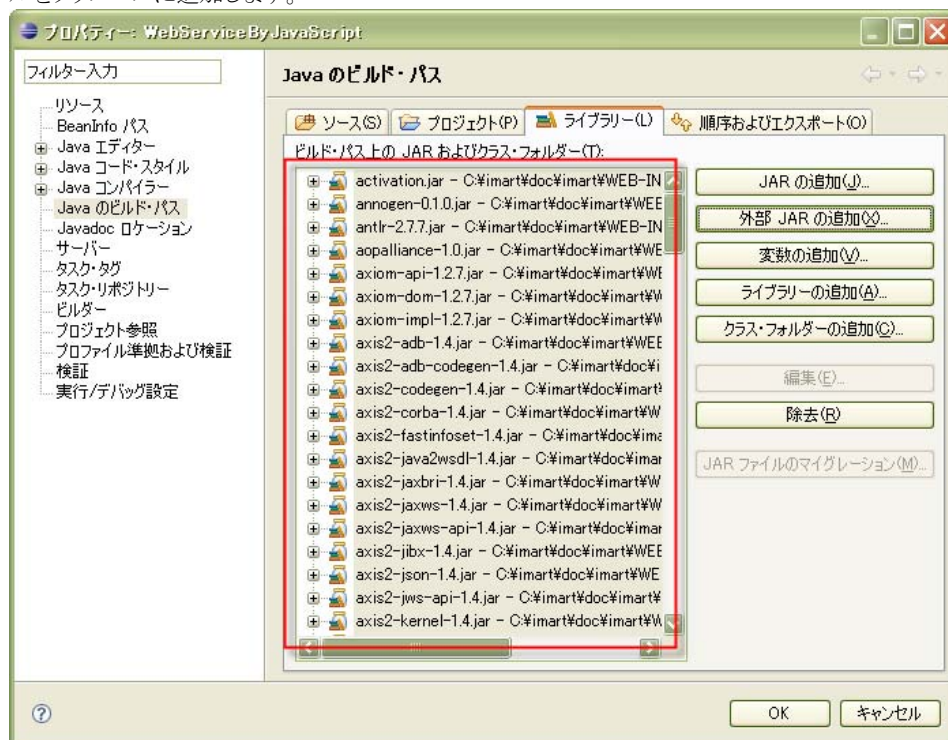
JavaScript 関数を実行するための JavaScript ラッパークラスを作成します。ここでは、member_info_operator.js の関数「add()」と「find()」と「findAll()」を実行する Java クラスを作成します。

4.1.2.4.1 ビルド・パスにライブラリを追加

「WebServiceByJavaScript」プロジェクトを右クリックし、[ビルド・パス]-[ビルド・パスの構成]を選択します。



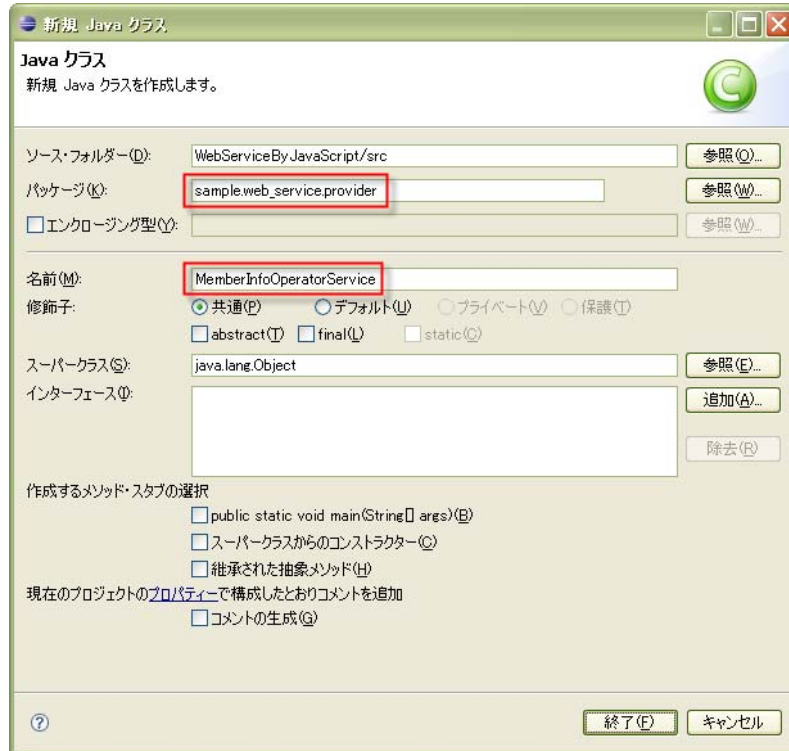
[ライブラリ]タブの [外部 JAR の追加] ボタンをクリックし、%IM_HOME%/doc/imart/WEB-INF/lib の Jar ファイルをクラスパスに追加します。



4.1.2.4.2 JavaScriptラッパークラスの新規作成

[パッケージ・エクスプローラ]ビューの「WebServiceByJavaScript」プロジェクト内の「src」ディレクトリ上で右クリックし、[新規]-[クラス]を選択します。

パッケージ名を「sample.web_service.provider」、クラス名を「MemberInfoOperatorService」と入力して、[終了]ボタンをクリックします。



MemberInfoOpeartorService.java が以下のように作成されます。

```
package sample.web_service.provider;

public class MemberInfoOperatorService {

}
```

4.1.2.4.3 JavaScript関数呼び出し処理の追加

JavaScript 関数呼び出し処理を追加します。

JavaScript 関数の呼び出しには、**jp.co.intra_mart.jssp.util.JavaScriptUtility** クラスの **executeFunction()** および **executeVoidFunction()** メソッドを利用します。このメソッドを利用することで、関数のパラメータが Java 形式から JavaScript 形式へ自動的に変換されます。同様に、関数の実行結果も、JavaScript 形式から Java 形式へ自動的に変換されます。

JavaScriptUtility#executeFunction()メソッドには、引数が 4 つあります。

第 1 引数「pagePath」は、実行する JavaScript ファイルのパスを指定します。(拡張子なし)

第 2 引数「functionName」は、実行する関数名を指定します。

第 3 引数「returnType」は、関数返却値の変換後の型(クラス)を指定します。

第 4 引数「args」は、実行する関数へのパラメータを指定します。(可変長引数)

変換後の Java クラスを配列で指定する場合、例えば、**Member** クラスの配列を変換後のクラスに指定する場合は以下ようになります。

```
(Member[]) JavaScriptUtility.executeFunction(pagePath, functionName, Member[].class, args);
```

JavaScriptUtility#executeVoidFunction()メソッドは、返却値の無い関数を実行する場合に利用します。

JavaScriptUtility#executeVoidFunction()メソッドには、引数が 3 つあります。

第 1 引数「pagePath」は、実行する JavaScript ファイルのパスを指定します。(拡張子なし)

第 2 引数「functionName」は、実行する関数名を指定します。

第 3 引数「args」は、実行する関数へのパラメータを指定します。(可変長引数)

JavaScriptUtility の詳細は、API リストを参照してください。

具体的には、MemberInfoOperatorService.java を以下のように編集します。(が追加した主な箇所です)

```

1: package sample.web_service.provider;
2:
3: import jp.co.intra_mart.foundation.web_service.auth.WSUserInfo;
4: import jp.co.intra_mart.jssp.util.JavaScriptUtility;
5:
6: import org.apache.axis2.AxisFault;
7:
8: public class MemberInfoOperatorService {
9:
10:     public Boolean add( WSUserInfo wsUserInfo, Member member ) throws AxisFault {
11:         try {
12:             String pagePath = "sample/web_service/provider/member_info_operator";
13:             String functionName = "add";
14:
15:             JavaScriptUtility.executeVoidFunction(pagePath,
16:                                                 functionName,
17:                                                 member);
18:
19:             return true;
20:         }
21:         catch (Exception ex) {
22:             throw AxisFault.makeFault(ex);
23:         }
24:     }
25:
26:     public Member find( WSUserInfo wsUserInfo, String id ) throws AxisFault {
27:         try {
28:             String pagePath = "sample/web_service/provider/member_info_operator";
29:             String functionName = "find";
30:
31:             Member member =
32:                 (Member) JavaScriptUtility.executeFunction(pagePath,
33:                                                         functionName,
34:                                                         Member.class,
35:                                                         id);
36:
37:             return member;
38:         }
39:         catch (Exception ex) {
40:             throw AxisFault.makeFault(ex);
41:         }
42:     }
43:
44:     public Member[] findAll( WSUserInfo wsUserInfo ) throws AxisFault {
45:         try {
46:             String pagePath = "sample/web_service/provider/member_info_operator";
47:             String functionName = "findAll";
48:
49:             Member[] members =
50:                 (Member[]) JavaScriptUtility.executeFunction(pagePath,
51:                                                            functionName,
52:                                                            Member[].class);
53:
54:             return members;
55:         }
56:         catch (Exception ex) {
57:             throw AxisFault.makeFault(ex);
58:         }
59:     }
60: }

```

- 各メソッドの第 1 引数には、Web サービス実行時の認証、および、認可に利用されるユーザ情報用引数を追加します。型を「`jp.co.intra_mart.foundation.web_service.authentication.WSUserInfo`」クラス、名称を「`wsUserInfo`」として追加します。（変数名の大文字・小文字は厳密に判定します）
- 引数「`wsUserInfo`」は、プログラム中では使用しないでください。（この変数は Web サービスの認証モジュール用です）ログインユーザの情報を取得する場合は、JavaScript 関数の中で、`AccessSecurityManager` を利用してください。
- 上記の 22, 40, 57 行目のように、「`throw AxisFault.makeFault(ex)`」を利用することで、JavaScript関数の中でスローしたSOAPFaultオブジェクトの内容をWebサービス・クライアントに返信することができます。なお、スクリプト開発モデルでSOAPフォルトを返信する方法は、「6.7 SOAPフォルトの送信方法」を参照してください。

4.1.2.5 Webサービスのデプロイ

4.1.2.5.1 jarファイルの作成

JavaScript ラッパークラス、および、型情報クラスを jar ファイルにまとめ、intra-mart へ反映します。
ここでは、「`sample_js_web_service.jar`」という名前の jar ファイルを作成します。

メニュー[ファイル]-[エクスポート]を選択します。

[JAR ファイル]を選択し、[次へ]ボタンをクリックします。



先ほど作成した「Member.java」と「MemberInfoOperatorService.java」をチェックします。

JAR ファイルのエクスポート先を「%IM_HOME%/doc/imart/WEB-INF/lib/sample_js_web_service.jar」に設定し
[終了]をクリックします。



4.1.2.5.2 aarファイルの作成

JavaScript ラッパークラスを Web サービスとして公開するために、aar ファイル (Axis2 Archive ファイル) を作成します。ここでは、aar ファイル作成用の Ant タスク「aarGenerate」を利用します。この Ant タスクを利用することで、intra-mart の認証・認可に必要な Axis2 モジュール「im_ws_auth」が適用された aar ファイルを作成することが可能です。

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インストールを行ってください。

- [Antのインストール](http://www.jajakarta.org/ant/ant-1.6.1/docs/ja/manual/install.html) (http://www.jajakarta.org/ant/ant-1.6.1/docs/ja/manual/install.html)

4.1.2.5.2.1 AarGen.xml の設定

%IM_HOME%/bin/tools/web_service/AarGen.xml の「Web サービス名(6 行目)」と「Web サービスとして公開する Java クラス名(9 行目)」を設定します。ここでは、「serviceName」を「SampleMemberInfoOperatorService」、className を「sample.web_service.provider.MemberInfoOperatorService」とします。

4.1.2.5.2.2 バッチファイルの実行

%IM_HOME%/bin/tools/AarGen.bat を実行します。

これにより、%IM_HOME%/doc/imart/WEB-INF/services/ディレクトリに「SampleMemberInfoOperatorService.aar」が作成されます。

4.1.2.6 アクセス権限の設定

次に、Webサービスを実行可能にするための権限設定を行います。「3.4 アクセス権限の設定 (16ページ)」を参考に設定を行ってください。

以上で、Web サービスのデプロイは完了です。

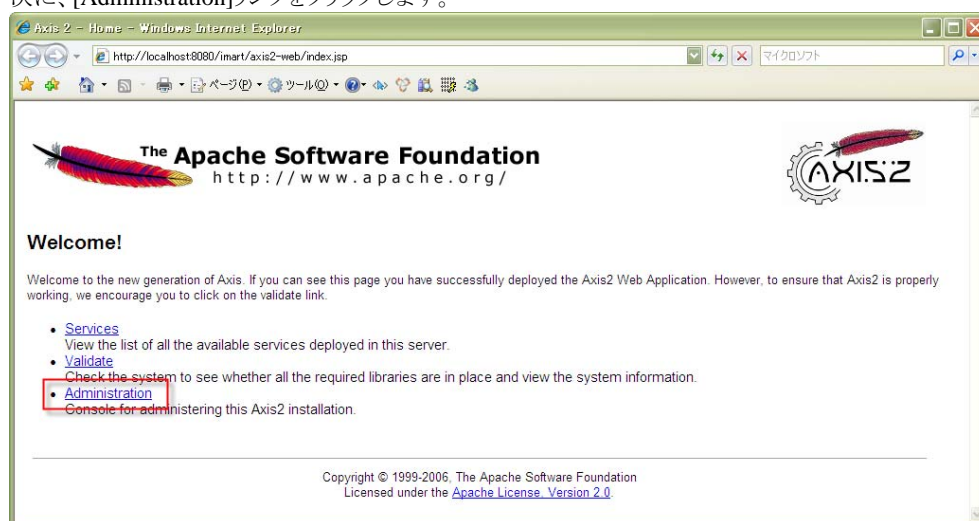
4.1.2.7 Webサービスがデプロイされていることを確認

Axis2 の管理コンソールを利用して、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。

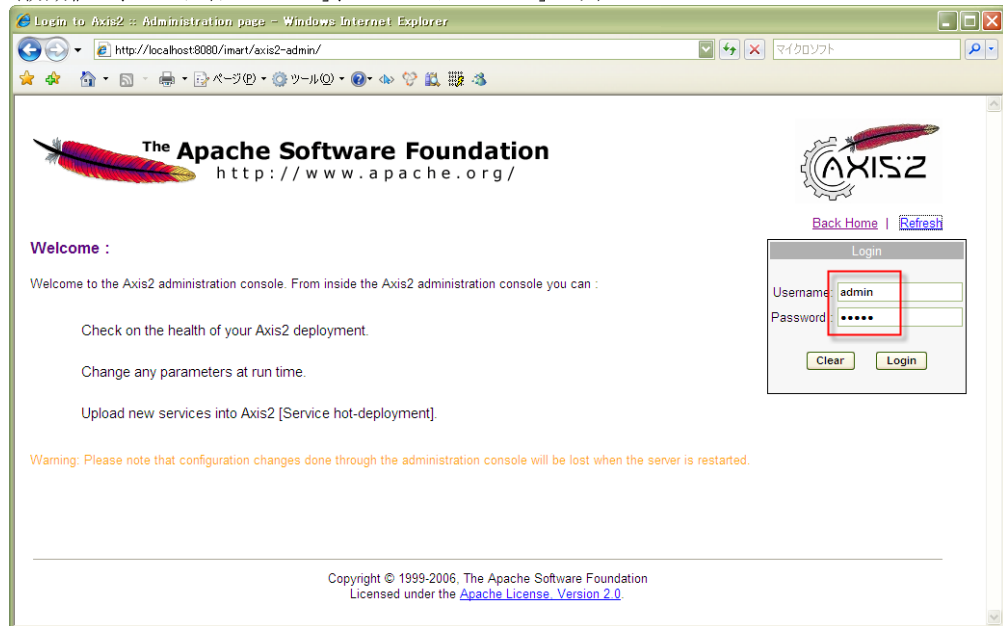
Axis2 の管理コンソールの詳細は、Axis2 プロジェクトの「[Apache Axis2 Web Administrator's Guide](http://ws.apache.org/axis2/1_4_1/webadminguide.html) (http://ws.apache.org/axis2/1_4_1/webadminguide.html)」ページを参照してください。

intra-martを起動し、Webブラウザから「<http://localhost:8080/imart/axis2-web/index.jsp>」にアクセスします。(ホスト名とポート番号は適宜読み替えてください)

次に、[Administration]リンクをクリックします。



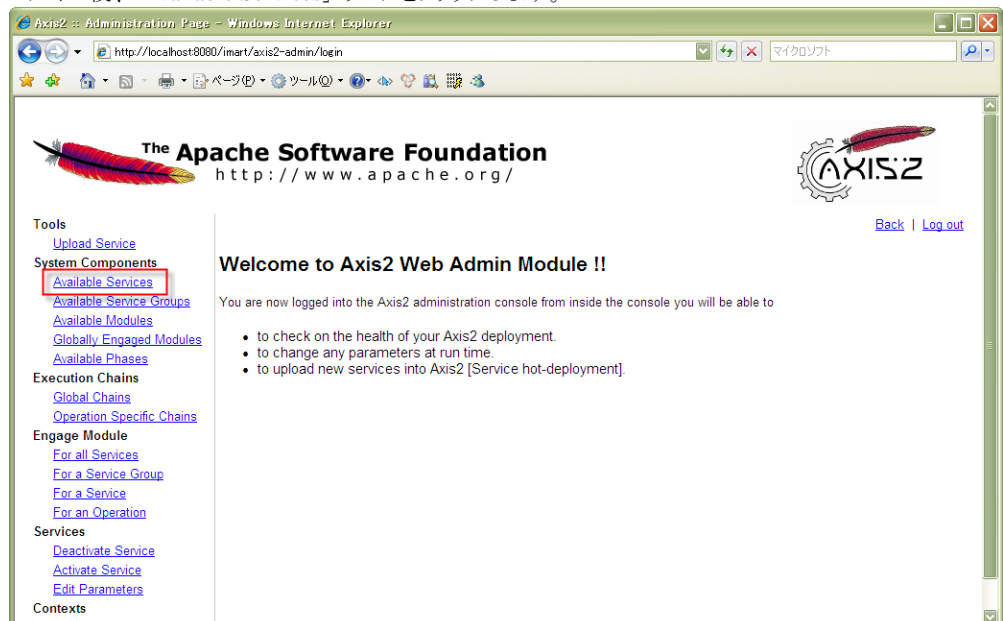
ユーザ名とパスワードを入力し、Axis2 の管理コンソールにログインします。
(初期値は、ユーザ名が「admin」、パスワードが「axis2」です)



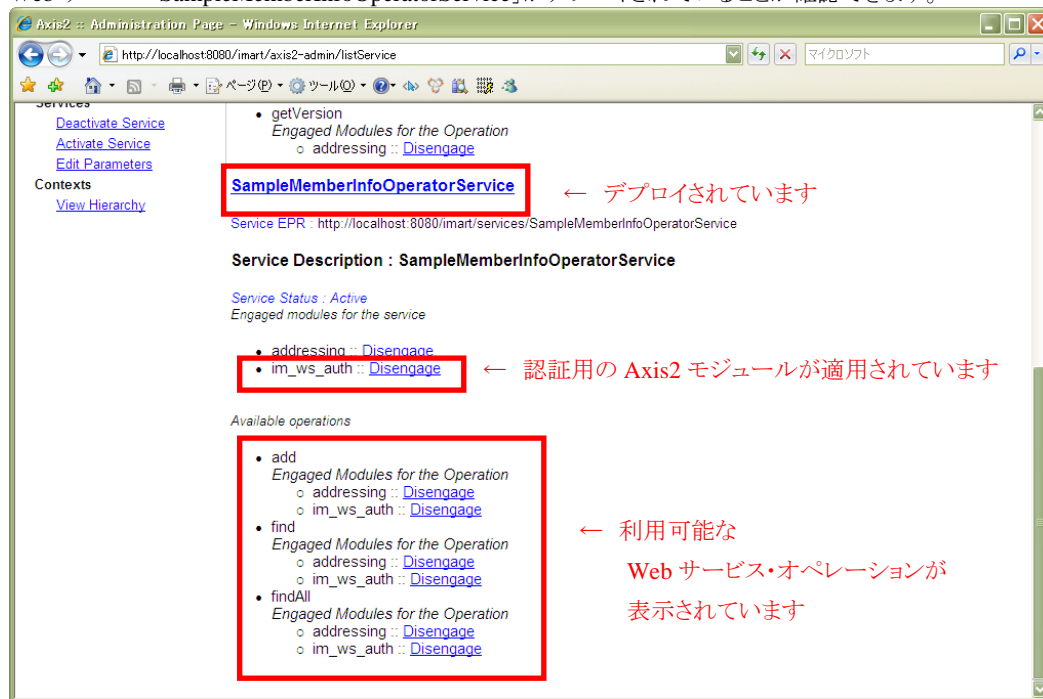
ユーザ名とパスワードは%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xml で設定可能です。

```
<axisconfig name="AxisJava2.0">
  .
  .
  .
  <parameter name="userName">admin</parameter>
  <parameter name="password">axis2</parameter>
  .
  .
</axisconfig>
```

ログイン後、「Available Services」リンクをクリックします。



Web サービス「SampleMemberInfoOperatorService」がデプロイされていることが確認できます。



なお、intra-mart の起動時に、以下のエラーが発生する場合は、Web サービスが重複しています。

「4.1.2.2.2.1 重複するサンプルの削除」を参考に、重複しているWebサービスを削除してください。

```
[ERROR] o.a.a.d.ServiceDeployer - The SampleMemberInfoOperatorService.aar service, which is not valid, caused Two services cannot have same name. A service with the SampleMemberInfoOperatorService name already exists in the system.
```

これで、Web サービスがデプロイされていることが確認できました。適宜 Web サービス・クライアントを作成して、Web サービスの動作を確認してください。

スクリプト開発モデルでWebサービス・クライアントを作成する方法は、「4.2 Webサービス・クライアントの作成」に記載されています。

なお、ここで説明した方法以外にAxis2 にWebサービスをデプロイする方法が「6.4 Webサービスのデプロイ方法」に記載されています。あわせてご参照ください。

4.2 Webサービス・クライアントの作成

4.2.1 概要

この章では、スクリプト開発モデルのファンクションコンテナから Web サービスを呼び出す手順を示します。

スクリプト開発モデルでは、Web サービスを呼び出すための API「**SOAPClient オブジェクト**」が用意されています。SOAPClient オブジェクトを利用することにより、XML や Java を意識することなく、Web サービスを呼び出すことが可能です。

4.2.2 詳細手順

SOAPClient オブジェクトを利用した 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 SOAPClient オブジェクトのインスタンスを生成
2. Web サービスを呼び出すソースコードのサンプルを表示
3. Web サービスの呼び出し（上記で出力された内容をカスタマイズ）

以降、サンプル「%IM_HOME%/pages/src/sample/web_service/client/member_info_operator_client.js」の「add()関数」を元に Web サービスが呼び出されるまでを解説します。

4.2.2.1 WSDLを指定して SOAPClientオブジェクト のインスタンスを生成

まず、SOAPClient オブジェクトのインスタンスを生成します。

```
1:  var wsdIFileURL = "http://localhost:8080/imart/services/SampleMemberInfoOperatorService?wsdl";
.  .
.  .
.  .
15: /**
16:  * メンバー情報を追加します。
17:  */
18: function add() {
19:
20:     //*****
21:     // ステップ 1 : WSDLを指定して SOAPClientオブジェクト のインスタンスを生成
22:     //*****
23:     try {
24:         var soapClient = new SOAPClient(wsdIFileURL);
25:         Debug.print("ステップ 1 完了");
26:     }
27:     catch(ex) {
28:         Debug.browse("エラーが発生しました。", ex);
29:     }
```

24 行目で SOAPClient オブジェクトのインスタンスを生成しています。コンストラクタの第 1 引数には、WSDL の URL を指定します。このサンプルでは、ローカルホストで公開されている Web サービス「SampleMemberInfoOperatorService」の WSDL を指定しています(1 行目)。インスタンス生成時に、不正な WSDL が指定された場合や対応していない WSDL が指定された場合は例外がスローされます。

SOAPClient は、インスタンス生成時に以下の処理を行います。

1. WSDL の解析
2. Web サービス・クライアントとなる Java スタブ・クラスのソース生成
3. Java スタブ・クラスのコンパイル
4. Java スタブ・クラスを呼び出す JavaScript ソースの生成

上記で示した一連の処理は、インスタンス生成時に一度だけ行われます。2 回目以降のインスタンス生成では上記処理が省略され、1 回目に作成されたスタブが利用されます。なお、この動作は、初期設定時の動作です。設定を変更する方法は「4.2.3 SOAPClientオブジェクトの設定」を参照してください。

上記の「WSDL の解析」、および、「Web サービス・クライアントとなる Java スタブ・クラスのソース生成」は Axis2 の CodeGenerationEngine を利用しています。そのため、Axis2 が対応していない WSDL が指定された場合はエラーが発生します。

SOAPClient オブジェクトは、Storage Service 上に保存されている WSDL ファイルを利用することが可能です。また、WSDL 内に複数の Web サービスが定義されている場合は、呼び出す Web サービス名を指定することも可能です。詳しくは API リストを参照してください。

4.2.2.2 Webサービス呼び出すソースコードのサンプルを表示

次に、Web サービス呼び出すためのソースコードを生成します。

```

32: //*****
33: // ステップ2 : Webサービス呼び出すソースコードのサンプルを表示
34: //*****
35: var sampleCode = soapClient.getSampleCode("add");
36: var msg = "ステップ2 完了。" + "\n";
37:     msg += "Webサービス呼び出すソースコードのサンプルが表示されました。" + "\n";
38:     msg += "pages/src/sample/web_service/client/member_info_operator_client.jsの35, 40行目を" + "\n";
39:     msg += "コメントアウトしてステップ3を実行してください";
40: Debug.browse(msg, sampleCode);

```

「getSampleCode()」を実行し、Web サービス呼び出すソースコードのサンプルを取得します。(35 行目)

getSampleCode() の引数には Web サービス・オペレーション名を指定します。オペレーション名は getOperationNames()でも取得可能です。引数を指定せずに getSampleCode() を実行すると、Web サービス内で利用可能なすべてのオペレーションに関するソースコードが返却されます。

getSampleCode()関数の目的はサンプルコードの生成です。したがって、getSampleCode()関数の実行ロジックは、Web サービス呼び出すコードの作成完了後に削除してください。

■ getSampleCode() が返却するソースコード例

```

1: *****
2: Usage:
3:     var result = soapClient.add(wsUserInfo, member);
4:
5: //-----
6: // Sample Data : 'wsUserInfo'
7: //-----
8: var wsUserInfo =
9: /* Object <WSUserInfo> */
10: {
11:     /* String */
12:     "password" : "prop_password",
13:
14:     /* String */
15:     "authType" : "prop_authType",
16:
17:     /* String */
18:     "userID" : "prop_userID",
19:
20:     /* String */
21:     "loginGroupID" : "prop_loginGroupID"
22: };
23:
24: //-----
25: // Sample Data : 'member'
26: //-----
27: var member =
28: /* Object <Member> */
29: {
30:     /* Boolean */
31:     "married" : true,
32:
33:     /* Number */
34:     "age" : 123,
35:

```

```
36:  /* String */
37:  "name" : "prop_name",
38:
39:  /* String */
40:  "id" : "prop_id",
41:
42:  /* Array <Member[]> */
43:  "children" : [
44:
45:  ],
46:
47:  /* Date (Thu Jun 19 2008 12:34:56 GMT+0900 (JST)) */
48:  "birthDate" : new Date(1213846496000)
49: };
50:
51: *****
```

サンプルコードは JSON 形式で出力されます。そのため、コピー＆ペーストで利用することが可能です。

また、コメントとして JavaScript の型情報が付与されています。9 行目の「<WSUserInfo>」や 28 行目の「<Member>」など、JavaScript の型情報の右側に表示されている「<>」内の文字列は、そのオブジェクト構成を表す名称です。例えば、42 行目の「<Member[]>」は、「children プロパティには、28 行目で示されている<Member>形式のオブジェクトが配列で格納される」ことを意味しています。

なお、XML スキーマの restriction で定義されている型などは、サンプルデータが生成されません。サンプルデータが生成されていない型については、WSDL、および、実行する Web サービスの仕様を確認してください。

4.2.2.3 Webサービスの呼び出し (getSampleCode()関数で取得した内容をカスタマイズ)

```

43: //*****
44: // ステップ3 : Webサービスの呼び出し (ステップ2で出力された内容をカスタマイズ)
45: //*****
46: // ↓↓↓↓ コピー&ペースト (ここから) ↓↓↓↓
47: //-----
48: // Sample Data : 'wsUserInfo'
49: //-----
50: var wsUserInfo =
51: /* Object <WSUserInfo> */
52: {
53:     /* String */
54:     "password" : WSAuthDigestGenerator4WSSE.getDigest(wsUserID, wsPassword) ,
55:
56:     /* String */
57:     "authType" : WSAuthDigestGenerator4WSSE.getAuthType(),
58:
59:     /* String */
60:     "userID" : wsUserID,
61:
62:     /* String */
63:     "loginGroupID" : wsLoginGroupID
64: };
65:
66: //-----
67: // Sample Data : 'member'
68: //-----
69: var member =
70: /* Object <Member> */
71: {
72:     /* Boolean */
73:     "married" : true,
74:
75:     /* Number */
76:     "age" : 123,
77:
78:     /* String */
79:     "name" : "prop_name",
80:
81:     /* String */
82:     "id" : "prop_id",
83:
84:     /* Array <Member[]> */
85:     "children" : [
86:
87:     ],
88:
89:     /* Date (Thu Jun 19 2008 12:34:56 GMT+0900 (JST)) */
90:     "birthDate" : new Date(1213846496000)
91: };
92: // ↑↑↑↑ コピー&ペースト (ここまで) ↑↑↑↑
93:
94:
95: //-----
96: // Webサービスの呼び出し
97: //-----
98: try{
99:     var result = soapClient.add(wsUserInfo, member);
100: }
101: catch (soapFault) {
102:     Debug.browse("エラーが発生しました。", soapFault);
103: }
104:
105: Debug.browse("ステップ3 完了",
106:     "追加しました。",
107:     "結果 : " + result);
108: }

```

getSampleCode()関数で取得したソースコードをカスタマイズし、Web サービスを呼び出します。

4.2.2.3.1 認証・認可用のユーザ情報の設定

まず、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」を利用することとします。認証タイプ「WSSE」の詳細は、「3.3.1.1 認証タイプ「WSSE」」を参照してください。

54 行目で、WSAuthDigestGenerator4WSSE オブジェクト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 オブジェクトは、そのパスワードのダイジェストの生成に特化したユーティリティです。この API は「ユーザ ID」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。

57 行目では、認証タイプ名を設定しています。これは文字列「WSSE」を設定することに相当します。

60 行目では、ユーザ ID を、63 行目では、ログイングループ ID を設定しています。

4.2.2.3.2 アプリケーション固有の設定

アプリケーション固有の設定、つまり、ビジネスロジックの実行に必要な情報を設定します。

このサンプルでは、69 行目から 91 行目で、登録するメンバー情報を設定しています。ここではコンソール上に出 forceされたサンプルコードをそのまま設定します。

4.2.2.3.3 Webサービス・オペレーションの実行

99 行目で Web サービス・オペレーションを呼び出します。

4.2.2.3.3.1 SOAP フォルトの受信方法

Web サービスの結果が SOAP フォルトとして返却された場合、スクリプト開発モデルでは、SOAPFault オブジェクトが例外としてスローされます。SOAPFault オブジェクトは、XML 形式の SOAP フォルトを JavaScript のオブジェクト形式に変換したものです。

SOAPFault オブジェクトを捕捉する(=Web サービス・オペレーションの実行ロジック部分を try/catch で囲む)ことで、SOAPFault オブジェクトを利用したエラー処理を行う事が可能となります。SOAPFault オブジェクトの詳細は、API リス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、「4.2.2.3.1 認証・認可用のユーザ情報の設定」で設定したユーザ情報を元に認証・認可が行われます。該当するユーザが存在しない、ログイングループが存在しない、パスワードが間違っている等のユーザ情報が不正な場合、または、Webサービス・オペレーションを実行する権限がない場合には SOAPFaultオブジェクトが例外としてスローされます。SOAPFaultオブジェクトの「faultCode」プロパティには、発生した問題に対応するコードが含まれています。コードの内容に関しては、「3.6 認証・認可機能のSOAPフォルト・コード一覧」を参照してください。

4.2.2.3.3.2 実行結果の確認

105 行目で Web サービス・オペレーションの実行結果を表示しています。

この Web サービス・オペレーションでは、実行結果が `true` で返却されていれば、メンバー情報が正しく登録されたことを意味しています。

これで、Web サービスの実行が完了しました。

サンプル「%IM_HOME%/pages/src/sample/web_service/client/member_info_operator_client.js」には、登録したメンバー情報を検索する「`findAll()`関数」も用意されています。実行して動作を確認してみてください。

なお、Web サービスの実行時にやり取りされている SOAP メッセージを閲覧する方法が「6.2 SOAP メッセージのモニタリング」に記述されています。あわせてご参照ください。

4.2.3 SOAPClientオブジェクトの設定

Application Runtime 上の conf/imart.xml にて、SOAPClient オブジェクトの各種設定が可能です。

imart.xml の intra-mart/platform/service/application/jssp タグに以下の設定が可能です。

設定項目	概要	初期値
soap-client/mode	<p>SOAPClient のインスタンス生成時に行われる Web サービスのスタブ生成(=WSDL の解析、Java スタブ・ソースの生成&コンパイル、および、JavaScriptソースの生成)に関する設定です。以下の3つの設定が可能です。</p> <p>・Everytime Web サービスのスタブを毎回作成します。開発時に利用する設定です。</p> <p>・Once Web サービスのスタブが存在しない場合のみスタブを作成します。</p> <p>・Never Web サービスのスタブを自動生成しません。このモードの場合、別途、Web サービスのスタブを配備する必要があります。スタブの配置には、Axis2 が生成する Java のスタブ・クラスをクラスパスに追加し、SOAPClient オブジェクトで生成された JavaScript のスタブ・ソースをソースディレクトリに追加する必要があります。</p> <p>サンプル「sample/web_service/client/member_info_operator_client.js」を使用し、soap-client/work-dir を変更してない場合に Web サービスのスタブを配備する方法の具体例を以下に示します。</p> <p>サンプル「sample/web_service/client/member_info_operator_client.js」を実行すると、「Axis2 が生成した Java のスタブ・クラス」および、「SOAPClient オブジェクトが生成した JavaScript のスタブ・ソース」が下記のディレクトリに出力されます。</p> <p>・Java のスタブ・クラス保存先 : %IM_HOME%/work/jssp/_functioncontainer/ja_JP/sample/web_service/provider/配下の*.class ファイル</p> <p>・JavaScript のスタブ・ソース保存先 : %IM_HOME%/work/jssp/_functioncontainer/ja_JP/sample/web_service/provider/配下の*.js ファイル</p> <p>・このディレクトリは、iWP を再起動すると削除されますので、ご注意ください。</p> <p>Axis2 が生成する Java のスタブ・クラスをクラスパスに追加するには、作成された Java のスタブ・クラスを下記にコピーしてください。</p> <p>・Java のスタブ・クラスコピー先 : %IM_HOME%/doc/imart/WEB-INF/classes/sample/web_service/provider/</p> <p>SOAPClient オブジェクトで生成された JavaScript のスタブ・ソースをソースディレクトリ(通常は%IM_HOME%/pages/src/)に追加するには、作成された JS のスタブ・ソースを下記にコピーしてください。</p> <p>・JavaScript のスタブ・ソースコピー先 : %IM_HOME%/pages/src/sample/web_service/provider/</p>	Once
soap-client/work-dir	<p>Web サービスのスタブ、および、WSDL ファイルを展開するディレクトリです。</p> <p>Application Runtime がインストールされているディレクトリからの相対パスで指定します。</p>	ファンクションコンテナの自動コンパイル時のクラスファイル出力先ディレクトリ
soap-client/javac-encoding	Web サービスのスタブをコンパイルする際の Java ソースの文字コードを設定します。	UTF-8
soap-client/javac-verbose	Web サービスのスタブをコンパイルする際の詳細情報出力可否設定です。true の場合、詳細情報が出力され、false の場合詳細情報は出力されません。	false

soap-client/javac-xmx	Web サービスのスタブをコンパイルする際の最大ヒープサイズを設定します。例えば、<javac-xmx>256m</javac-xmx>と設定した場合、最大ヒープサイズは 256M バイトに設定されます。	利用する JavaVM の -Xmx オプションの初期値
soap-client/wsdl/storage/import-location/suffixes/suffix	スタブ生成に必要なファイルの拡張子を設定します。Storage Service 上に保存されている WSDL ファイルを利用する際に必要な設定です。 例えば、ある WSDL ファイルで参照している要素が、別のファイルで定義されている場合、その定義ファイルの拡張子をここに設定します。	「.xsd」と「.wsdl」
soap-client/wsdl/storage/import-location/sub-dirs/sub-dir	スタブ生成に必要なファイルが格納されているディレクトリ名の設定です。Storage Service 上に保存されている WSDL ファイルを利用する際に必要な設定です。 例えば、ある WSDL ファイルで参照している要素が、別のファイルで定義されている場合、そのファイルが保存されているディレクトリ名をここに設定します。WSDL ファイルが保存されているディレクトリのサブディレクトリ名として利用されます。	「xsd」

以下に imart.xml の設定例を示します。

```

<intra-mart>
  <platform>
    <service>
      <application enable="true">
        .
        .
        .
      </application>
      <jssp>
        .
        .
        .
      </jssp>
      <soap-client>
        <mode>Once</mode>
        <work-dir>work/jssp/_functioncontainer</work-dir>
        <javac-encoding>UTF-8</javac-encoding>
        <javac-verbose>>false</javac-verbose>
        <javac-xmx>64m</javac-xmx>
        <wsdl>
          <storage>
            <import-location>
              <suffixes>
                <suffix>.xsd</suffix>
                <suffix>.wsdl</suffix>
              </suffixes>
              <sub-dirs>
                <sub-dir>xsd</sub-dir>
              </sub-dirs>
            </import-location>
          </storage>
        </wsdl>
      </soap-client>
    </jssp>
  </platform>
  .
  .
  .
</intra-mart>

```

4.2.4 WebLogicでSOAPClientオブジェクトを利用する方法

WebLogic で SOAPClient オブジェクトを利用する場合、別途 AXIS2 のダウンロード、および、環境変数「AXIS2_HOME」を設定する必要があります。

4.2.4.1 Axis2 のダウンロード

Axis2 1.4 をダウンロードします。具体的には、http://ws.apache.org/axis2/download/1_4_1/download.cgiにアクセスし、「Standard Binary Distribution」のzipファイルをダウンロードします。ダウンロードが完了したら、「axis2-1.4.1-bin.zip」を任意のディレクトリに解凍します。

4.2.4.2 環境変数「AXIS2_HOME」の設定

次に、環境変数「AXIS2_HOME」を設定します。「axis2-1.4.1-bin.zip」を解凍したディレクトリを、環境変数「AXIS2_HOME」として設定してください。「%AXIS2_HOME%/lib」が、Axis2 の各種 Jar ファイルが格納されているディレクトリを指し示すように設定を行ってください。

4.2.5 WSDLがhttpsで提供されている場合のSOAPClient利用方法

WSDL が https で提供されている場合の SOAPClient 利用方法について説明します。例として、ここでは、下記 URL で定義された Web サービスを SOAPClient で利用する方法を示します。

- `https://hostname/imart/services/SampleWebService?wsdl`

4.2.5.1 サーバ証明書の取得

まず、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 FireFox3 を利用して証明書を取得する方法を示します。
(サーバ証明書の詳細は、サーバ管理者にお問い合わせください)

1. メニューより、[ツール] - [オプション]を選択します。
2. [詳細]ペインの[暗号化]タブを表示します。
3. [証明書を表示]ボタンを押下し、[証明書マネージャ]ウィンドウを表示します。
4. [サーバ証明書]タブを表示し、取得したいサーバ証明書を選択します。
5. [表示]ボタンをクリックし、取得したい証明書であることを確認します。
6. 取得したい証明書であることを確認後、[エクスポート]ボタンを押下し、証明書を保存します。
(ア) ここでは、証明書を「**C:%temp%server.crt**」に保存します。

4.2.5.2 サーバ証明書をキーストアに追加

JDK 付属のツール keytool の -import コマンドを使って、信頼できる証明書をリストに追加します。
例えば、サーバ証明書「**C:%temp%server.crt**」を、別名「**sample_alias**」でキーストアエントリに格納するには、以下のコマンドを実行します。

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

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、JDKのドキュメントに含まれる [keytool - 鍵と証明書の管理ツール](#) を参照してください)

4.2.5.3 システムプロパティ「javax.net.ssl.trustStore」の設定

Application Runtime の imart.xml を編集します。imart.xml の intra-mart/platform/java/server/command/option にシステムプロパティ「javax.net.ssl.trustStore」を追加します。

具体的には、以下を追加します。(Windows の場合の例です。「user_name」は適宜変更してください)

```
-Djavax.net.ssl.trustStore="C:¥Documents and Settings¥user_name¥.keystore"
```

以上の設定を行うことにより、SSL 経由の Web サービスを利用することが可能です。ただし、WSDL の URL が「https://・・・」で提供されていても、WSDL に記述されているエンドポイントが、「https://・・・」では無い場合があるのでご注意ください。もし、エンドポイントが「https://・・・」では無い場合、明示的にエンドポイントを指定してください。

以下に、JavaScriptAPI「SOAPClient」の利用例を記述します。

```
var wsdlUrl      = "https://hostname/imart/services/SampleWebService?wsdl";
var serviceName = null;
var portName     = null;
var endpoint     = "https://hostname/imart/services/SampleWebService"; // ← 明示的に指定します。

var soapClient = new SOAPClient(wsdlUrl, serviceName, portName, endpoint);
.
.
.
(以降、通常のプログラムです)
```

4.2.6 FAQ

4.2.6.1 「指定した要求に失敗しました」が発生します

指定された Web サービス・オペレーションを実行する権限がない可能性があります。

詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.7 wsse: RequestFailed - 指定した要求に失敗しました」を参照してください。

4.2.6.2 「指定された RequestSecurityToken を理解できません」が発生します

認証タイプに対応する認証モジュールが存在しない場合に発生します。詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.2 wsse: BadRequest - 指定された RequestSecurityToken を理解できません」を参照してください。

4.2.6.3 「要求が無効か、形式が間違っています」が発生します

SOAP ボディにユーザ情報が存在しない、または、ユーザ情報が格納されている要素名が「wsUserInfo」ではない場合に発生します。

Webサービスとして公開するJavaクラス(=JavaScriptラッパークラス)のコンパイル方法が間違っている場合にも発生します。「6.2 SOAPメッセージのモニタリング」を参考にSOAPリクエスト内のユーザ情報を確認してください。

■ SOAP リクエスト抜粋

```
<soapenv:Body>
  <ns3:add xmlns:ns3="http://provider.web_service.sample">
    <ns3:param0>
      <ns1:authType
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        PlainTextPassword
      </ns1:authType>
      <ns1:loginGroupID
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        default
      </ns1:loginGroupID>
      <ns1:password
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        パスワード
      </ns1:password>
      <ns1:userID xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        ueda
      </ns1:userID>
    </ns3:param0>
    <ns3:param1>
      <ns2:age xmlns:ns2="http://provider.web_service.sample/xsd">123.0</ns2:age>
      <ns2:id xmlns:ns2="http://provider.web_service.sample/xsd">prop_id</ns2:id>
      <ns2:married xmlns:ns2="http://provider.web_service.sample/xsd">true</ns2:married>
      <ns2:name xmlns:ns2="http://provider.web_service.sample/xsd">prop_name</ns2:name>
    </ns3:param1>
  </ns3:add>
</soapenv:Body>
```

上記のように「param0」等になっている場合は、コンパイル方法が間違っています。

詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.1 wsse: InvalidRequest - 要求が無効か、形式が間違っています」を参照してください。

4.2.6.4 Webサービス化したJavaScript関数内でログインセッションが取得できません

Axis2 モジュール「im_ws_auth」が適用されていない可能性があります。Axis2 モジュールの適用方法に関しては「6.5 Axis2 モジュールの適用方法」を参照してください。

4.2.6.5 Storage Service上のWSDLファイルを利用するには？

Storage Service 上に保存されている WSDL ファイルを利用するには、その WSDL ファイルを指し示している VirtualFile オブジェクトを、SOAPClient オブジェクトのコンストラクタ第 1 引数に指定してください。

なお、WSDL ファイルの解析時に、別の WSDL ファイル や 別の XML スキーマファイル(以降、「XSD ファイル」と呼ぶ)が必要な場合は、コンストラクタに指定した WSDL ファイルと同じディレクトリ(または、設定可能なサブディレクトリ)にそれらのファイルを保存してください。

上記は、SOAP メッセージの送受信時に使われる要素が、指定した WSDL ファイル内ではなく、別の XSD ファイルで定義されている場合が当てはまります。

WSDLファイルの解析時に必要なファイルの拡張子や、必要なファイルが格納されているディレクトリ名の設定方法は、「4.2.3 SOAPClientオブジェクトの設定」を参照してください。

4.2.6.6 複数Webサービスが定義されているWSDLを利用するには？

WSDL 内に複数の Web サービスが定義されている場合は、Web サービス名を指定して SOAPClient オブジェクトを利用する必要があります。具体的には、SOAPClient オブジェクトのコンストラクタ第 2 引数に実行したい Web サービス名を指定します。

4.2.6.7 JavaScript形式からJava形式へのオブジェクト変換に失敗します

JavaScriptUtility#jsToJavaBean()でエラーが発生している場合は、JavaScript形式からJava形式へのオブジェクト変換に失敗しています。「4.1.1.1.2 プロパティ変換規則 : JavaScript形式 → Java形式」に違反していないかを確認してください。

以下に、エラーの具体例を示します。

4.2.6.7.1 **IllegalConversionException: Cannot convert 'JavaScript NativeArray' into 'Java class <クラス名>'**

以下のエラーは、「JavaScript の配列」を、「配列として定義されていないJavaBeanのプロパティ」に変換しようとした際に発生します。

```
IllegalConversionException: Cannot convert 'JavaScript NativeArray' into 'Java class <java.lang.String>'.  
(<java.lang.String> may not be declared as an array)
```

この場合の解決方法は、JavaScript 配列ではなく文字列を返却するか、JavaBean のプロパティを配列として定義する必要があります。

4.2.6.7.2 **NumberFormatException: For input string: "Xxxx" (文字列)**

以下のエラーは、「JavaScript の"aaaa"という文字列」から「java.lang.Number で定義されているJavaBeanのプロパティ」への変換時、数値変換に失敗した際に発生します。

```
java.lang.NumberFormatException: For input string: "aaaa"
```

この場合の解決方法は、JavaScript 側で文字列ではなく、数値を返却するか、JavaBean のプロパティをjava.lang.Stringとして定義する必要があります。

5 チュートリアル(JavaEE 開発モデル編)

5.1 Webサービス・プロバイダの作成

この章では、im-JavaEE Framework のサブフレームワークである、イベントフレームワークで構築されたアプリケーションを Web サービスとして公開する手順を記します。

この方法によりイベントフレームワークで作成された既存のアプリケーションを Web サービスとして再利用可能です。

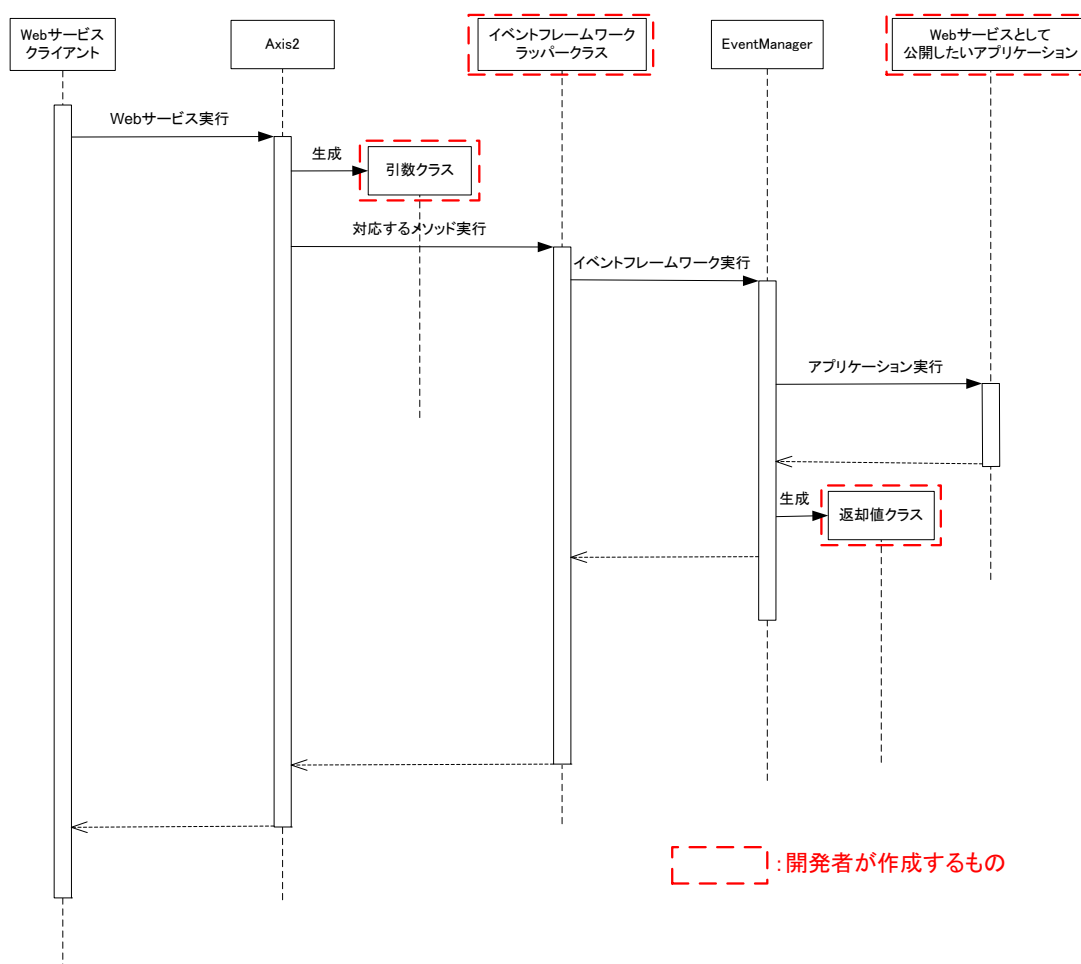
この章を読み進める上での注意点は以下の通りです。

- Java クラスの作成に Eclipse を利用します。
JavaEE 開発モデルのプログラム開発支援ツールである「intra-mart eBuilder」も利用可能です。
- 一般的な Java の知識が必要です。
Web サービス・プロバイダ及び、Web サービス・クライアントを作成するために一般的な Java の知識が必要です。
- im-JavaEE Framework の知識が必要です。
この章のサンプルプログラムでは、Web サービス・プロバイダから im-JavaEE Framework のサブフレームワークである、イベントフレームワークを利用して作成されたアプリケーションを呼び出します。im-JavaEE Framework の詳しい仕様については、「intra-mart WebPlatform/AppFramework im-JavaEE Framework 仕様書」を参照してください。

5.1.1 概要

イベントフレームワークで作成されたアプリケーションの Web サービス化は、イベントフレームワークで作成されたアプリケーションを実行するためのクラスを作成し、そのクラスを Web サービスとして公開することで実現します。
(以降、このクラスを「イベントフレームワークラッパークラス」と呼びます)

Web サービスとして公開されたアプリケーションが実行されるまでの流れを以下に示します。



- クライアントが、Web サービスの実行を要求します。
- Web サービス実行エンジン「Axis2」が、受け付けたリクエストに該当するイベントフレームワークラッパークラスのメソッドを呼び出します。
- ラッパークラスは「EventManager」を利用してアプリケーションを実行します。
- 実行結果がクライアントに返却されます。

開発者は、上記のシーケンス図の中で太い点線で囲まれたものを作成する必要があります。

一つ目は、「Web サービスとして公開したいイベントフレームで構築されたアプリケーション」です。二つ目は、イベントフレームワークを実行するための「イベントフレームワークラッパークラス」です。

これらに加えて、Web サービスへのリクエスト、クライアントへのレスポンスの形式によって、「引数クラス、返却値クラス」が必要になります。

5.1.2 詳細手順

この章では、イベントフレームワークを利用して作成されたアプリケーションを Web サービスとして公開する手順を示します。

5.1.2.1 Webサービス公開までの流れ

イベントフレームワークを利用して作成されたアプリケーションを Web サービスとして公開するまでの流れは以下の通りです。

1. 準備
 - Web サービスとして公開するアプリケーションの選定
 - intra-mart のインストール
 - Eclipse のインストール
2. アプリケーションのデプロイ
3. 引数クラス、返却値クラスの作成
4. イベントフレームワークラッパークラスの作成
5. Web サービスのデプロイ
 - jar ファイルの作成
 - aar ファイルの作成
6. アクセス権限の設定

5.1.2.2 準備

5.1.2.2.1 Webサービスとして公開するアプリケーションの選定

ここでは、サンプルとして以下の java クラス及びイベントコンフィグファイルを用意します。

EventListener は本来トランザクション処理などを行いますが、このサンプルは EventResult に静的な値をに格納する単純なプログラムです。

- foo/conf/event-config-bar.xml
- foo.model.event.FooEvent.java
- foo.model.event.FooEventListener.java
- foo.model.event.FooEventResult.java

イベントコンフィグファイルに定義されている、アプリケーション ID「foo.conf.bar」、イベントキー「do_event」に関連付けられている処理を Web サービスとして公開します。

このアプリケーションの引数(Event クラス)は以下の属性を保持しています。

プロパティ	説明 (型)
itemNo	商品番号 (Integer)

返却値(EventResult クラス)は以下の属性を保持します。

プロパティ	説明 (型)
itemNo	商品番号 (java.lang.Integer)
itemName	商品名 (java.lang.String)
valid	有効・無効 (java.lang.Boolean)
date	日付 (java.lang.Boolean)

以下にアプリケーションのソースファイルを記します。

foo/conf/event-config-bar.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<event-config>
  <event-group>
    <event-key>do_event</event-key>
    <event-class>foo.model.event.FooEvent</event-class>
    <event-factory>
      <factory-class>
        jp.co.intra_mart.framework.base.event.StandardEventListenerFactory
      </factory-class>
      <init-param>
        <param-name>listener</param-name>
        <param-value>foo.model.event.FooEventListener</param-value>
      </init-param>
    </event-factory>
  </event-group>
</event-config>
```

foo.model.event.FooEvent.java

```
package foo.model.event;

import jp.co.intra_mart.framework.base.event.Event;

public class FooEvent extends Event {

    private Integer itemNo;

    public Integer getItemNo() {
        return itemNo;
    }

    public void setItemNo(Integer itemNo) {
        this.itemNo = itemNo;
    }
}
```

foo.model.event.FooEventListener.java

```

package foo.model.event;

import java.util.Date;

import jp.co.intra_mart.framework.base.event.Event;
import jp.co.intra_mart.framework.base.event.EventResult;
import jp.co.intra_mart.framework.base.event.StandardEventListener;
import jp.co.intra_mart.framework.system.exception.ApplicationException;
import jp.co.intra_mart.framework.system.exception.SystemException;

public class FooEventListener extends StandardEventListener {

    protected EventResult fire(Event arg) throws SystemException,
        ApplicationException {

        FooEvent event = (FooEvent)arg;
        FooEventResult result = new FooEventResult();
        result.setItemNo(event.getItemNo());
        result.setItemName("商品A");
        result.setValid(Boolean.TRUE);
        result.setDate(new Date());
        return result;
    }
}

```

foo.model.event.FooEventResult.java

```

package foo.model.event;

import java.util.Date;

import jp.co.intra_mart.framework.base.event.EventResult;

public class FooEventResult implements EventResult {

    private Integer itemNo;
    private String itemName;
    private Boolean valid;
    private Date date;

    public Integer getItemNo() {
        return itemNo;
    }
    public void setItemNo(Integer itemNo) {
        this.itemNo = itemNo;
    }
    public String getItemName() {
        return itemName;
    }
    public void setItemName(String itemName) {
        this.itemName = itemName;
    }
    public Boolean getValid() {
        return valid;
    }
    public void setValid(Boolean valid) {
        this.valid = valid;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
}

```

5.1.2.2.2 intra-martのインストール

intra-mart をインストールします。詳しくは、intra-mart のセットアップガイドを参照してください。

この章では、intra-mart WebPlatform (Resin)がスタンドアローンでインストールされている事とします。

以降、intra-mart がインストールされているディレクトリを `%IM_HOME%` とします。

5.1.2.2.3 Eclipseのインストール

この章では、Java のクラスを作成するために Eclipse を利用します。Eclipse とは、オープンソースの統合ソフトウェア開発環境(IDE)です。なお、Eclipse ではなく、スクリプト開発モデルおよびJavaEE 開発モデルのプログラム開発支援ツールである「**intra-mart eBuilder**」を利用することも可能です。

[Eclipse.orgのダウンロードページ\(http://www.eclipse.org/downloads/index.php\)](http://www.eclipse.org/downloads/index.php) からEclipseをダウンロードします。

ここでは、「Eclipse IDE for java EE Developers」をダウンロードします。



ここでは、「eclipse-jee-europa-winter-win32.zip」をダウンロードした事とします。

ダウンロードしたアーカイブを解凍します。以降、この解凍先のパスを「`%ECLIPSE_HOME%`」とします。

日本語化が必要な場合は、Webサイト「[エクリプスWiki \(http://eclipsewiki.net/eclipse/index.php\)](http://eclipsewiki.net/eclipse/index.php)」内の [日本語化プラグイン](#) ページを参考に日本語化を行うと良いでしょう。この章では、Pleiadesを利用して日本語化を行った Eclipse を利用します。

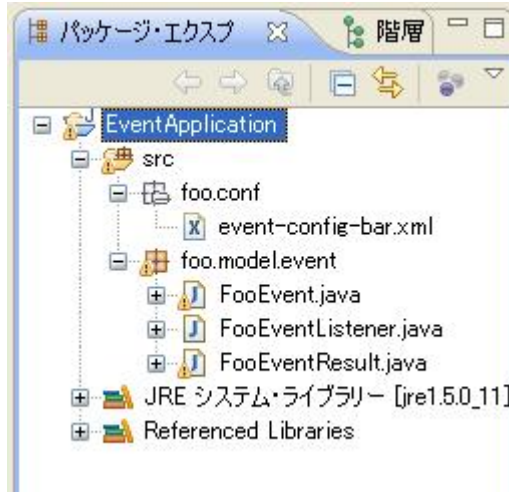
5.1.2.3 アプリケーションのデプロイ

イベントフレームワークで作成されたアプリケーションを intra-mart に配置します。

5.1.2.3.1 ソースファイルの読み込み

Eclipse を起動し、新規に Java プロジェクトを作成します。

ここでのプロジェクト名は「**EventApplication**」とし、プロジェクト内に以下のようなパッケージ構成で、「5.1.2.2.1 Webサービスとして公開するアプリケーションの選定」に記したJavaソースとコンフィグファイルを配置してください。



これらの Java ソースをコンパイルするためには%IM_HOME%/doc/imart/WEB-INF/lib 配下の全ての jar ファイルを、java プロジェクトのビルドパスに割り当てる必要があります。

5.1.2.3.2 jarファイルの作成

メニュー[ファイル]-[エクスポート]を選択します。

[JAR ファイル]を選択し、[次へ]ボタンをクリックします。

プロジェクト内の全ての Java ソースとコンフィグファイルをチェックし、エクスポート先を「%IM_HOME%/doc/imart/WEB-INF/lib/event_application.jar」と設定してエクスポートを行ってください。

以上でアプリケーションの配置は完了です。

5.1.2.4 返却値クラスの作成

Web サービス・プロバイダは商品情報を Web サービス・クライアントに送信します。ここでは商品情報を格納し、Web サービス・クライアントに送信するためのモデルクラスを作成します。

新規に Java プロジェクトを作成します。

ここでのプロジェクト名は「**WebServiceByJavaEE**」とします。

プロジェクト作成時に %IM_HOME%/doc/imart/WEB-INF/lib 配下の全ての jar ファイルおよび、%IM_HOME%/bin/httpd/jsdk-15.jar を java プロジェクトのビルドパスに割り当ててください。

以下の「sample.web_service.provider.Item.java」を作成してください。

```
package sample.web_service.provider;

import java.util.Date;

public class Item {
    private Integer itemNo;
    private String itemName;
    private Boolean valid;
    private Date date;
    public Integer getItemNo() {
        return itemNo;
    }
    public void setItemNo(Integer itemNo) {
        this.itemNo = itemNo;
    }
    public String getItemName() {
        return itemName;
    }
    public void setItemName(String itemName) {
        this.itemName = itemName;
    }
    public Boolean getValid() {
        return valid;
    }
    public void setValid(Boolean valid) {
        this.valid = valid;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
}
```

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、継承関係を持ったクラスを指定することはできません。利用した場合、Web サービス・クライアント側でエラーとなる場合があります。Web サービス・クライアントとして、Axis2 の Stub を利用している場合、「ADBException: Unexpected subelement XXXX(=要素名)」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という、ADB (Axis Data Binding) の現行仕様による制限です。

5.1.2.5 イベントフレームワークラッパークラスの作成

以下の「sample.web_service.provider.EventService.java」を作成してください。

```
package sample.web_service.provider;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import jp.co.intra_mart.common.aid.jsdk.javax.servlet.http.HTTPContext;
import jp.co.intra_mart.common.aid.jsdk.javax.servlet.http.HTTPContextManager;
import jp.co.intra_mart.foundation.web_service.auth.WSUserInfo;
import jp.co.intra_mart.framework.base.event.EventManager;
import jp.co.intra_mart.framework.base.util.UserInfo;
import jp.co.intra_mart.framework.base.util.UserInfoUtil;

import org.apache.axis2.AxisFault;

import foo.model.event.FooEvent;
import foo.model.event.FooEventResult;

public class EventService {

    public Item doEvent(WSUserInfo wsUserInfo, Integer itemNo)
        throws AxisFault {

        try {
            HTTPContext ctx =
                HTTPContextManager.getInstance().getCurrentContext();
            HttpServletRequest req = ctx.getRequest();
            HttpServletResponse res = ctx.getResponse();
            UserInfo user = UserInfoUtil.createUserInfo(req, res);

            String appID = "foo.conf.bar";
            String eventID = "do_event";
            EventManager manager = EventManager.getEventManager();
            FooEvent event = (FooEvent)manager.createEvent(appID, eventID, user);
            event.setItemNo(itemNo);
            FooEventResult result = (FooEventResult)manager.dispatch(event);

            Item item = new Item();
            item.setItemNo(result.getItemNo());
            item.setItemName(result.getItemName());
            item.setValid(result.getValid());
            item.setDate(result.getDate());
            return item;

        } catch (Exception ex) {
            throw AxisFault.makeFault(ex);
        }
    }
}
```

イベントフレームワークラッパークラスでは、EventResult の値を Item クラスのインスタンスに入れ替えて返却しています。このように EventResult をそのまま返却するのではなく Web サービスの返却値クラスを利用することで Web サービスの処理結果の形式を明確化することができます。

5.1.2.6 Webサービスのデプロイ

5.1.2.6.1 jarファイルの作成

イベントフレームワークラッパークラス及び、返却値クラスを jar ファイルにまとめ、intra-mart へ反映します。
ここでは、「sample_javaee_web_service.jar」という名前の jar ファイルを作成します。

メニュー[ファイル]-[エクスポート]を選択します。
[JAR ファイル]を選択し、[次へ]ボタンをクリックします。

先ほど作成した「Item.java」と「EventService.java」をチェックし、エクスポート先を「%IM_HOME%/doc/imart/WEB-INF/lib/sample_javaee_web_service.jar」と設定してエクスポートを行ってください。

5.1.2.6.2 aarファイルの作成

イベントフレームワークラッパークラスを Web サービスとして公開するために、aar ファイル (Axis2 Archive ファイル)を作成します。ここでは、aarファイル作成用の Antタスク「aarGenerate」を利用します。この Antタスクを利用することで、intra-mart の認証・認可に必要な Axis2 モジュール「im_ws_auth」が適用された aar ファイルを作成することが可能です。

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インストールを行ってください。

- [Antのインストール](http://www.jajakarta.org/ant/ant-1.6.1/docs/ja/manual/install.html) (http://www.jajakarta.org/ant/ant-1.6.1/docs/ja/manual/install.html)

5.1.2.6.2.1 AarGen.xml の設定

%IM_HOME%/bin/tools/web_service/AarGen.xml の「Web サービス名(6 行目)」と「Web サービスとして公開する Java クラス名(9 行目)」を設定します。ここでは、「serviceName」を「EventService」、className を「sample.web_service.provider.EventService」とします。

5.1.2.6.2.2 バッチファイルの実行

%IM_HOME%/bin/tools/AarGen.bat を実行します。
これにより、%IM_HOME%/doc/imart/WEB-INF/services/ディレクトリに「EventService.aar」が作成されます。

5.1.2.7 アクセス権限の設定

次に、Webサービスを実行可能にするための権限設定を行います。「3.4 アクセス権限の設定 (16ページ)」を参考に設定を行ってください。

以上で、Web サービスのデプロイは完了です。

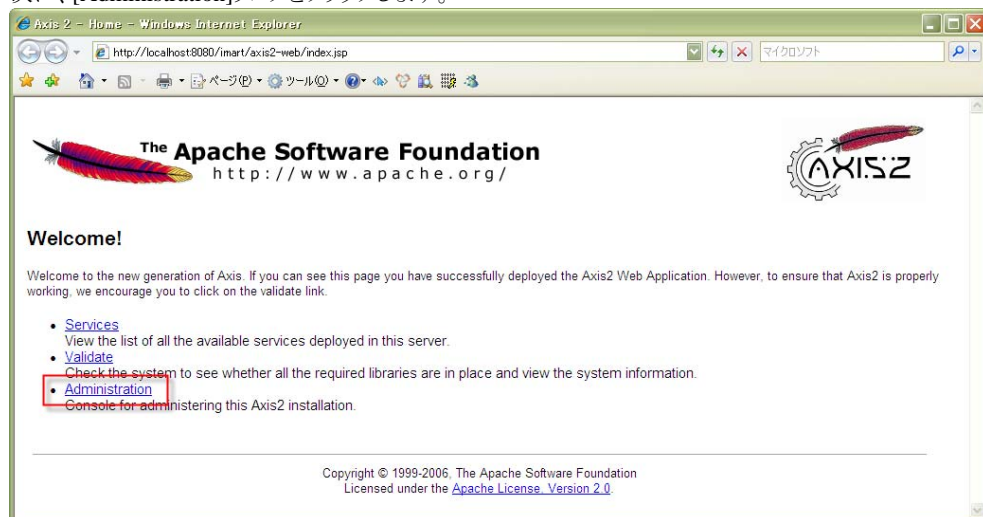
5.1.2.8 Webサービスがデプロイされていることを確認

Axis2 の管理コンソールを利用して、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。

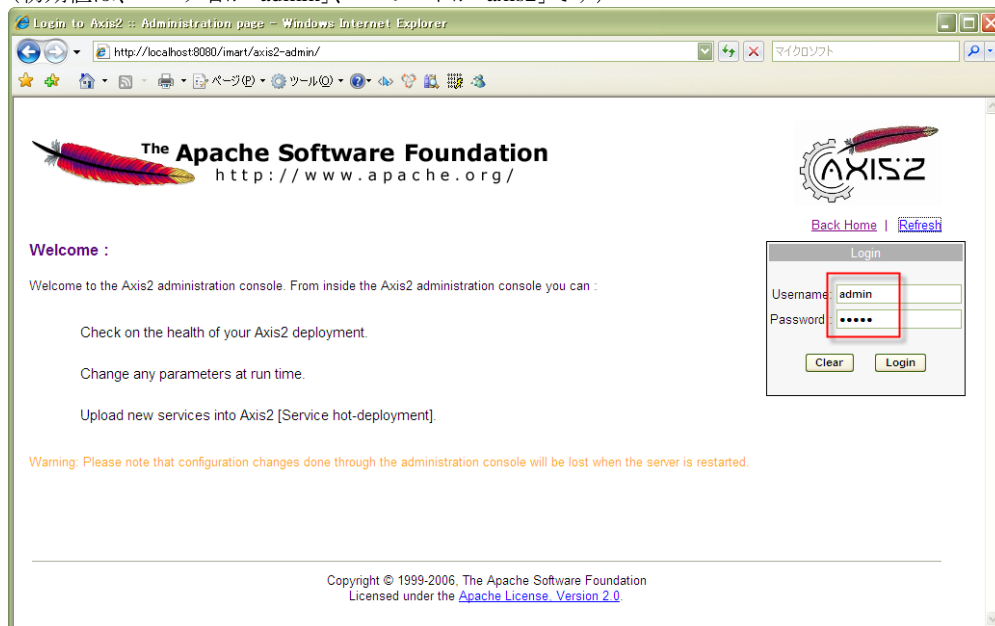
Axis2 の管理コンソールの詳細は、Axis2 プロジェクトの「[Apache Axis2 Web Administrator's Guide](http://ws.apache.org/axis2/1_4_1/webadminguide.html) (http://ws.apache.org/axis2/1_4_1/webadminguide.html)」ページを参照してください。

intra-martを起動し、Webブラウザから「<http://localhost:8080/imart/axis2-web/index.jsp>」にアクセスします。(ホスト名とポート番号は適宜読み替えてください)

次に、[Administration]リンクをクリックします。



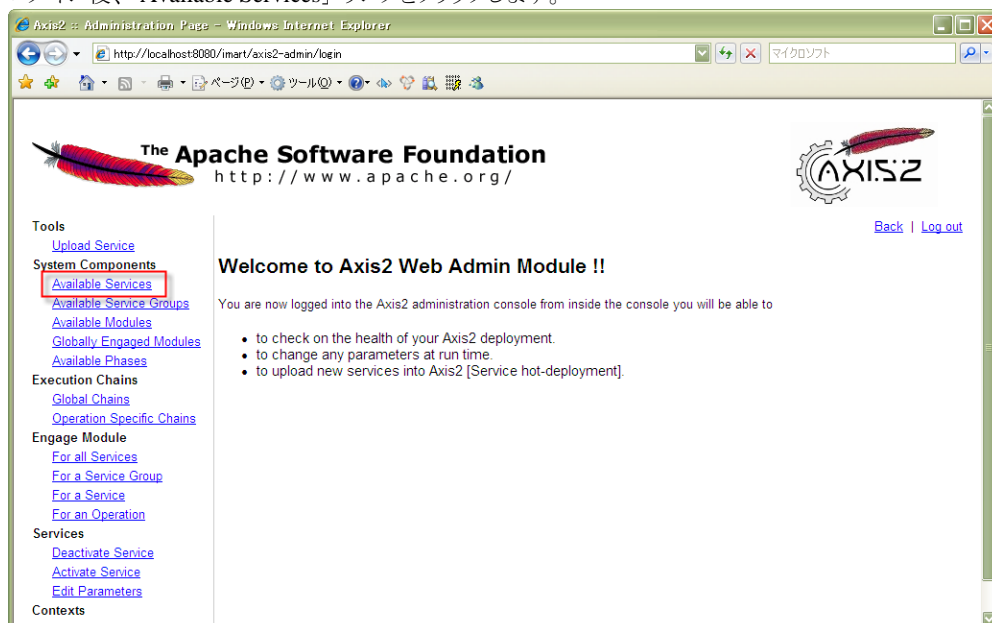
ユーザ名とパスワードを入力し、Axis2 の管理コンソールにログインします。
(初期値は、ユーザ名が「admin」、パスワードが「axis2」です)



ユーザ名とパスワードは%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xml で設定可能です。

```
<axisconfig name="AxisJava2.0">
  .
  .
  .
  <parameter name="userName">admin</parameter>
  <parameter name="password">axis2</parameter>
  .
  .
</axisconfig>
```

ログイン後、「Available Services」リンクをクリックします。



Web サービス「EventService」がデプロイされていることが確認できます。



これで、Web サービスがデプロイされていることが確認できました。適宜 Web サービス・クライアントを作成して、Web サービスの動作を確認してください。

JavaEE開発モデルでWebサービス・クライアントを作成する方法は、「5.2 Webサービス・クライアントの作成」に記載されています。

なお、ここで説明した方法以外にAxis2 にWebサービスをデプロイする方法が「6.4 Webサービスのデプロイ方法」に記載されています。あわせてご参照ください。

5.2 Webサービス・クライアントの作成

5.2.1 概要

この章では、Java クラスから Web サービスを呼び出す手順を示します。

JavaEE 開発モデルでは、Web サービスを呼び出すためのスタブ・クラスを生成する Ant タスクである AntCodegenTask を利用して Web サービスを呼び出すことが可能です。

5.2.2 詳細手順

AntCodegenTask 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
2. Web サービスを呼び出す実行クラスを作成
3. Web サービスの呼び出し

以降、イベントフレームワークで作成されたサンプルアプリケーションを元に Web サービスが呼び出されるまでを解説します。

5.2.2.1 WSDLを指定してスタブクラスを生成

ここでは、スタブ作成用の Ant タスク「AntCodegenTask」を利用してスタブ・クラスがアーカイブされた jar ファイルを生成します。

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インストールを行ってください。

- [Antのインストール](http://www.jajakarta.org/ant/ant-1.6.1/docs/ja/manual/install.html) (http://www.jajakarta.org/ant/ant-1.6.1/docs/ja/manual/install.html)

5.2.2.1.1 StubGen.xml の設定

%IM_HOME%/bin/tools/web_service/StubGen.xml の「WSDL のファイルパス、または URL(7 行目)」と「jar ファイル名(41 行目)」を設定します。

ここでは、「wsdlfilename」を「http://localhost:8080/imart/services/EventService?wsdl」、「jarfile」を「\${dest.dir}/event_stub.jar」とします。

WSDL が https で提供されている場合は、WSDL ファイルをローカル環境にダウンロード後、そのファイルを StubGen.xml の「WSDL のファイルパス、または URL(7 行目)」に指定してください。

5.2.2.1.2 バッチファイルの実行

intra-mart を起動し、%IM_HOME%/bin/tools/StubGen.bat を実行します。

これにより、%IM_HOME%/bin/tools/web_service/stub/ディレクトリに「event_stub.jar」が作成されます。

5.2.2.2 Webサービスを呼び出す実装クラスの作成

Eclipse を起動し、新規に Java プロジェクトを作成します。ここでのプロジェクト名は「WebServiceClientByJavaEE」とします。プロジェクト作成時に%IM_HOME%/doc/imart/WEB-INF/lib 配下の全ての jar ファイルおよび、先ほど作成した%IM_HOME%/bin/tools/web_service/stub/ event_stub.jar を java プロジェクトのビルドパスに割り当ててください。

以下の「sample.web_service.provider.ClientMain.java」を作成してください。

```
package sample.web_service.provider;

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

public class ClientMain {

    public static void main(String[] args) {

        try {
            // スタブを生成
            String endPoint = "http://localhost:8080/imart/services/EventService";
            EventServiceStub stub = new EventServiceStub(endPoint);

            // ユーザ情報
            String loginGroupID = "default";
            String userID = "guest";
            String password = "guest";

            // WSSE認証用のパスワードダイジェストを生成
            WSAuthDigestGenerator4WSSE gen = new WSAuthDigestGenerator4WSSE();
            String passwordDigest = gen.getDigest(loginGroupID, userID, password);

            // ログイン認証用のユーザ情報を生成
            EventServiceStub.WSUserInfo wsUserInfo = new EventServiceStub.WSUserInfo();
            wsUserInfo.setLoginGroupID(loginGroupID);
            wsUserInfo.setUserID(userID);
            wsUserInfo.setPassword(passwordDigest);
            wsUserInfo.setAuthType(gen.getAuthType());

            // パラメータを設定し、Webサービスを呼び出す
            EventServiceStub.DoEvent param = new EventServiceStub.DoEvent();
            param.setWsUserInfo(wsUserInfo);
            param.setItemNo(1234);
            EventServiceStub.DoEventResponse res = stub.doEvent(param);
            EventServiceStub.Item item = res.get_return();

            // 処理結果を出力
            System.out.println(item.getItemNo());
            System.out.println(item.getItemName());
            System.out.println(item.getValid());
            System.out.println(item.getDate());

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

5.2.2.3 Webサービスの呼び出し

パッケージエクスプローラから「ClientMain.java」を右クリックし、「実行」→「Java アプリケーション」を選択してください。実行結果がコンソールに出力されます。

5.2.2.3.1 SSL経由のWebサービスを利用する場合

システムプロパティ「javax.net.ssl.trustStore」を設定することで、SSL経由のWebサービスを利用することができます。「4.2.5 WSDLがhttpsで提供されている場合のSOAPClient利用方法」と同様の手順で利用可能となります。あわせてご参照ください。

5.2.3 FAQ

5.2.3.1 「指定した要求に失敗しました」が発生します

指定された Web サービス・オペレーションを実行する権限がない可能性があります。

詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.7 wsse: RequestFailed - 指定した要求に失敗しました」を参照してください。

5.2.3.2 「指定された RequestSecurityToken を理解できません」が発生します

認証タイプに対応する認証モジュールが存在しない場合に発生します。詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.2 wsse:BadRequest - 指定された RequestSecurityToken を理解できません」を参照してください。

5.2.3.3 「要求が無効か、形式が間違っています」が発生します

SOAP ボディにユーザ情報が存在しない、または、ユーザ情報が格納されている要素名が「wsUserInfo」ではない場合に発生します。

Webサービスとして公開するJavaクラス(=イベントフレームワークラッパークラス)のコンパイル方法が間違っている場合にも発生します。「6.2 SOAPメッセージのモニタリング」を参考にSOAPリクエスト内のユーザ情報を確認してください。

■ SOAP リクエスト抜粋

```
<soapenv:Body>
  <ns3:add xmlns:ns3="http://provider.web_service.sample">
    <ns3:param0>
      <ns1:authType
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        PlainTextPassword
      </ns1:authType>
      <ns1:loginGroupID
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        default
      </ns1:loginGroupID>
      <ns1:password
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        パスワード
      </ns1:password>
      <ns1:userID xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        ueda
      </ns1:userID>
    </ns3:param0>
    <ns3:param1>
      <ns2:age xmlns:ns2="http://provider.web_service.sample/xsd">123.0</ns2:age>
      <ns2:id xmlns:ns2="http://provider.web_service.sample/xsd">prop_id</ns2:id>
      <ns2:married xmlns:ns2="http://provider.web_service.sample/xsd">true</ns2:married>
      <ns2:name xmlns:ns2="http://provider.web_service.sample/xsd">prop_name</ns2:name>
    </ns3:param1>
  </ns3:add>
</soapenv:Body>
```

上記のように「param0」等になっている場合は、コンパイル方法が間違っています。

詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.1 wsse:InvalidRequest - 要求が無効か、形式が間違っています」を参照してください。

5.2.3.4 Webサービス化したイベントフレームワーク内でログインセッションが取得できません

Axis2 モジュール「im_ws_auth」が適用されていない可能性があります。Axis2 モジュールの適用方法に関しては「6.5 Axis2 モジュールの適用方法」を参照してください。

6 開発時に有用な情報

6.1 発生しやすいエラーについて

6.1.1 「指定した要求に失敗しました」が発生します

指定された Web サービス・オペレーションを実行する権限がない可能性があります。

詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.7 wsse: RequestFailed - 指定した要求に失敗しました」を参照してください。

6.1.2 「指定された RequestSecurityToken を理解できません」が発生します

認証タイプに対応する認証モジュールが存在しない場合に発生します。認証タイプが未設定の際に発生する場合があります。詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.2 wsse: BadRequest - 指定された RequestSecurityToken を理解できません」を参照してください。

6.1.3 「要求が無効か、形式が間違っています」が発生します

SOAP ボディにユーザ情報が存在しない、または、ユーザ情報が格納されている要素名が「wsUserInfo」ではない場合に発生します。

Webサービスとして公開するJavaクラス(=JavaScriptラッパークラス)のコンパイル方法が間違っている場合にも発生します。「6.2 SOAPメッセージのモニタリング」を参考にSOAPリクエスト内のユーザ情報を確認してください。

■ SOAP リクエスト抜粋

```
<soapenv:Body>
  <ns3:add xmlns:ns3="http://provider.web_service.sample">
    <ns3:param0>
      <ns1:authType
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        PlainTextPassword
      </ns1:authType>
      <ns1:loginGroupID
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        default
      </ns1:loginGroupID>
      <ns1:password
        xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        パスワード
      </ns1:password>
      <ns1:userID xmlns:ns1="http://auth.web_service.foundation.intra_mart.co.jp/xsd">
        ueda
      </ns1:userID>
    </ns3:param0>
    <ns3:param1>
      <ns2:age xmlns:ns2="http://provider.web_service.sample/xsd">123.0</ns2:age>
      <ns2:id xmlns:ns2="http://provider.web_service.sample/xsd">prop_id</ns2:id>
      <ns2:married xmlns:ns2="http://provider.web_service.sample/xsd">true</ns2:married>
      <ns2:name xmlns:ns2="http://provider.web_service.sample/xsd">prop_name</ns2:name>
    </ns3:param1>
  </ns3:add>
</soapenv:Body>
```

上記のように「**param0**」等になっている場合は、コンパイル方法が間違っています。

詳しくは、「3.6 認証・認可機能のSOAPフォルト・コード一覧」の「3.6.1 wsse:InvalidRequest - 要求が無効か、形式が間違っています」を参照してください。

6.1.4 Webサービス・プロバイダでログインセッションが取得できません

Axis2 モジュール「im_ws_auth」が適用されていない可能性があります。Axis2 モジュールの適用方法に関しては「6.5 Axis2 モジュールの適用方法」を参照してください。

6.2 SOAPメッセージのモニタリング

Web サービス実行時にやり取りされている SOAP メッセージをモニタリングする方法を紹介します。

6.2.1 SOAPMonitor

Axis2 に含まれている SOAP メッセージメッセージモニタリングツール「SOAPMonitor」の利用方法を示します。

SOAPMonitor はアプレットとして提供されているため、Web サーバと Application Runtime が別のマシンで稼動している場合は利用できません。(アプレットは設置されたウェブサイト以外との通信を許さないため)

SOAPMonitorの詳細は、Axis2 プロジェクトの [Using the SOAP Monitor](#) ページを参照してください。

6.2.1.1 web.xmlの編集

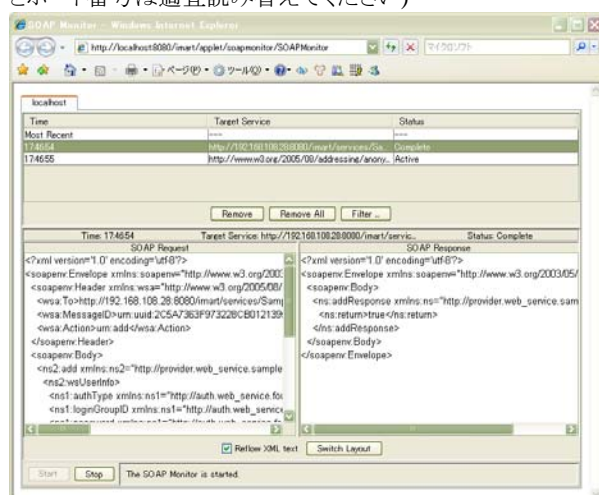
%IM_HOME%/doc/imart/WEB-INF/web.xml を編集します。初期状態はコメントアウトされている SOAPMonitorService サークレットの定義(1813 行目付近) と サークレットマッピングの定義(2019 行目付近)を有効にしてください。

6.2.1.2 axis2.xmlの編集

%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xmlを編集します。初期状態はコメントアウトされている Axis2 モジュールの設定「<module ref="soapmonitor"/>」(385 行目付近)を有効にしてください。

6.2.1.3 SOAPMonitorの起動

Web ブラウザより「<http://localhost:8080/imart/applet/soapmonitor/SOAPMonitor>」にアクセスしてください。(ホスト名とポート番号は適宜読み替えてください)



左側に SOAP リクエスト、右側に SOAP レスポンスの内容が表示されます。

6.2.2 TCPMon

Apache Web Services Project で提供されているメッセージモニタリングツール「TCPMon」の利用方法を示します。

ここでは、「4.2 Webサービス・クライアントの作成」のサンプル実行時、実際にやり取りされているSOAPメッセージをTCPMonで閲覧する方法を説明します。

TCPMonの詳細は、[TCPMonのプロジェクトページ](#) を参照してください。

6.2.2.1 TCPMonのダウンロード

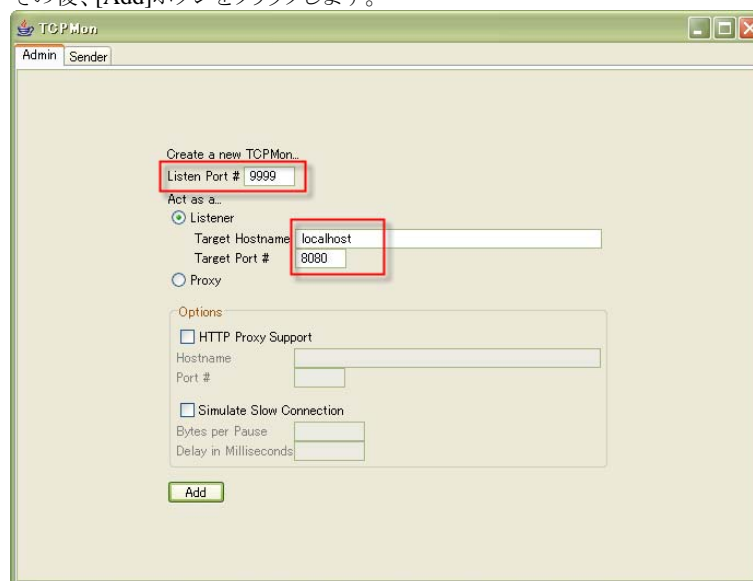
[TCPMonのダウンロードページ](http://ws.apache.org/commons/tcpmon/download.cgi) (<http://ws.apache.org/commons/tcpmon/download.cgi>)からTCPMonをダウンロードし、アーカイブを解凍します。このディレクトリを%TCPMON_HOME%とします。

6.2.2.2 TCPMonの起動と設定

%TCPMON_HOME%/build/tcpmon.bat (または tcpmon.sh) を実行し TCPMon を起動します。

[Admin]タブを表示し、Listen Port を「9999」に設定します。次に、Target Hostname と Target port に Web サービスがデプロイされているホストの情報を設定します。ここでは、それぞれ「localhost」、「8080」に設定します。

その後、[Add]ボタンをクリックします。



6.2.2.3 SOAPClientのエンドポイントを指定

「4.2 Webサービス・クライアントの作成」のサンプルを修正します。

SOAPClient オブジェクトのコンストラクタの第 4 引数に Web サービスのエンドポイントを指定します。これを TCPMon 経由の URL に設定します。ここでは、先ほど設定した TCPMon の Listen Port 「9999」を指定します。具体的には以下のようにソースを変更します。(下線が変更した箇所です)

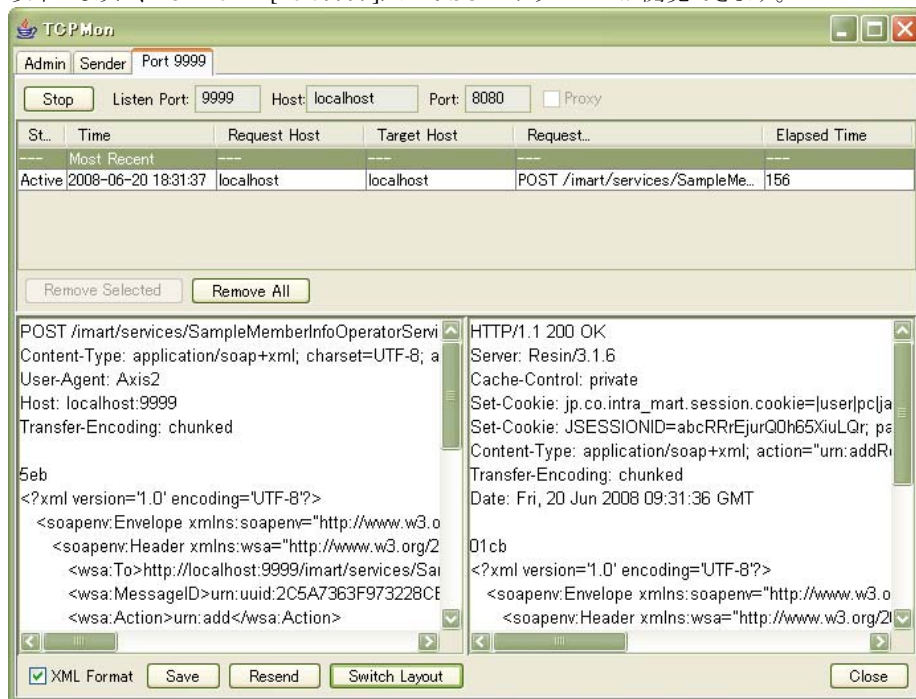
```

1:  var wsdlFileURL = "http://localhost:8080/imart/services/SampleMemberInfoOperatorService?wsdl";
2:  var endPoint = "http://localhost:9999/imart/services/SampleMemberInfoOperatorService";
3:  .
4:  .
5:  .
6:  .
15: /**
16:  * メンバー情報を追加します。
17:  */
18: function add() {
19:
20:     //*****
21:     // ステップ 1 : WSDLを指定して SOAPClientオブジェクト のインスタンスを生成
22:     //*****
23:     try {
24:         var soapClient = new SOAPClient(wsdlFileURL, null, null, endPoint);
25:         Debug.print("ステップ 1 完了");
26:     }
27:     catch(ex) {
28:         Debug.browse("エラーが発生しました。", ex);
29:     }

```

変更を保存後、上記サンプルを実行します。

以下のように、TCPMon の[Port 9999]タブで SOAP メッセージが閲覧できます。



6.3 バイナリファイルの送受信方法

Web サービスとして公開するメソッドの引数、および、返却値の型に「**byte[]**」を指定することで、バイナリファイルを受信、および、送信することが可能です。自動的にバイト配列が **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。

バイナリファイルの送受信方法を示したサンプルが収録されています。このサンプルは、Storage Service へのファイルアップロード、および、ダウンロードを行うサンプルです。Java ソースコードは、本製品 CD-ROM の公開ソースファイル「im_sample-src.zip」に含まれています。ご参照ください。

- Web サービス・プロバイダ側.
 - %IM_HOME%/doc/imart/WEB-INF/services/im_ws_auth_sample/META-INF/services.xml
 - %IM_HOME%/doc/imart/WEB-INF/classes/sample/web_service/provider/VirtualFileAccessService.class
 - %IM_HOME%/pages/src/sample/web_service/provider/virtual_file_access.js
- Web サービス・クライアント側
 - %IM_HOME%/pages/src/sample/web_service/client/virtual_file_access_client.js

6.3.1 バイナリファイル送受信時の注意点

Web サービスとして公開するメソッドの引数に **JavaBean** が指定されている場合、その **JavaBean** 内の「バイト配列(=byte[])」形式のプロパティは、データが正しく送受信されません。これは、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メソッドに「バイト配列(=byte[])」形式の引数を指定してください。

6.4 Webサービスのデプロイ方法

Web サービスのデプロイ方法を以下に説明します。

6.4.1 services.xmlについて

Axis2 には、Web サービス名や Web サービスとして公開する Java クラス名を指定するための設定ファイル「services.xml」があります。このファイルを決められたディレクトリ構成に配置したり、aar ファイルに含めたりすることで Web サービスのデプロイが可能となります。

■ services.xml のサンプル

```

1: <?xml version="1.0" encoding="UTF-8"?>
2: <service name="SampleMemberInfoOperatorService" >
3:   <Description>
4:     This is MemberInfoOperatorService (JavaScript Wrapper Class)
5:   </Description>
6:
7:   <parameter name="ServiceClass">sample.web_service.provider.MemberInfoOperatorService</parameter>
8:
9:   <module ref="im_ws_auth" />
10:
11:   <messageReceivers>
12:     <messageReceiver
13:       mep="http://www.w3.org/2004/08/wsdl/in-only"
14:       class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
15:     <messageReceiver
16:       mep="http://www.w3.org/2004/08/wsdl/in-out"
17:       class="org.apache.axis2.rpc.receivers.RPCMessageReceiver" />
18:   </messageReceivers>
19: </service>

```

行番号	説明
2 行目	Web サービス名が指定されています
3～5 行目	この Web サービスの説明が記述されています
7 行目	Web サービスとして公開する Java のクラス名を指定しています。 ここで指定されたクラスの public メソッド が Web サービス・オペレーションとして公開されます。
9 行目	Axis2 モジュール「im_ws_auth」を適用する設定が行われています。 これにより、この Web サービスは intra-mart の認証・認可の対象となります。(逆にこのモジュールを適用しない場合、認証・認可処理は 行われません)
12～14 行目	メッセージレシーバの設定です。ここでは、返却値のないメソッドに対して「RPCInOnlyMessageReceiver」を適用しています。
15～18 行目	メッセージレシーバの設定です。ここでは、何らかの返却を行うメソッドに対して「RPCMessageReceiver」を適用しています。

メッセージレシーバ「RPCxxxxMessageReceiver」は、XML の SOAP メッセージと Java のオブジェクトへのマッピングを行うためのものです。このほかに、「RawXMLxxxxMessageReceiver」という XML を直接扱うためのメッセージレシーバも用意されています。

services.xmlの詳細は、Axis2 プロジェクトの [Service Configuration](#) を参照してください。

メッセージレシーバの詳細は、Axis2 の API リストを参照してください。

6.4.2 ディレクトリ形式のデプロイ方法

%IM_HOME%/doc/imart/WEB-INF/services/配下に、任意の名称でディレクトリを作成すると Web サービスをデプロイすることが可能です。

「services.xml」を含めたパスは以下の通りです。

```
%IM_HOME%/doc/imart/WEB-INF/services/任意の名称で作成されたディレクトリ/META-INF/services.xml
```

Web サービスの認証・認可を行うために、Axis2 モジュール「im_ws_auth」を適用してください。

設定例として、%IM_HOME%/doc/imart/WEB-INF/services/im_ws_auth_sample/META-INF/services.xml が用意されています。

なお、「4.1 Webサービス・プロバイダの作成 - 4.1.2.5.2 aarファイルの作成」の

SampleMemberInfoOperatorService.aar ファイルにも services.xml は含まれています。aar ファイルは zip 形式で圧縮されていますので、適宜解凍しサンプルとしてご利用ください。

6.4.3 aarファイル形式のデプロイ方法

aar ファイル (Axis2 Archive ファイル) を、%IM_HOME%/doc/imart/WEB-INF/services ディレクトリにコピーすることで、Web サービスをデプロイすることが可能です。

aarファイルとは、Webサービスに関連するライブラリ、クラス、プロパティ・ファイル、WSDLファイル、XSDファイル、および、設定ファイル「services.xml」を格納したファイルです。aarファイルの構成は、Axis2 プロジェクトの「[Advanced User's Guide](#) - Step 3: Create Archive File」をご参照ください。

Axis2の管理コンソールを利用して、aarファイルをアップロードすることも可能です。詳しくは、Axis2プロジェクトの「[Apache Axis2 Web Administrator's Guide](http://ws.apache.org/axis2/1_4_1webadminguide.html) (http://ws.apache.org/axis2/1_4_1webadminguide.html)」ページを参照してください。

6.4.3.1 AntAarGenerateTask

services.xml を含んだ aar ファイルを作成する Ant タスクです。

指定された出力先ディレクトリに「Web サービス名.aar」という名称でファイルが作成されます。

属性	説明	必須
serviceName	Web サービス名	Yes
className	Web サービスとして公開する Java クラス名	Yes
destDir	出力先ディレクトリ	No、初期値は「この Ant タスク実行時のカレントディレクトリ」
moduleRef	Web サービスに適用する Axis2 モジュール名 複数指定する場合は「,(カンマ)」で区切ってください。	No、指定しない場合、Axis2 モジュールの適用は行われません。
wsdlDir	aar ファイルに含める WSDL 等のファイルが保存されているディレクトリ。 ここで指定されたディレクトリ直下のファイルを、すべて、services.xml と同じ階層に圧縮します。	No、指定しない場合は、services.xml 以外のファイル圧縮は行われません。
verbose	この Ant タスク実行時の詳細情報を出力します。	No、初期値は「false」

この Ant タスクを利用したビルドファイルのサンプルが「%IM_HOME%/bin/tools/web_service/AarGen.xml」に用意されています。bat ファイルや sh ファイルとあわせてご利用ください。

なお、この Ant タスクは、Jar ファイルや Java クラスを aar ファイルに含める機能はありません。そのため、この Ant タスクで生成された aar ファイルの Web サービスを利用するには、Web サービスの実行に必要な Jar ファイルや Java クラスが Web アプリケーション内で利用可能となっている必要があります。具体的には、Jar ファイルを %IM_HOME%/doc/imart/WEB-INF/lib ディレクトリに、Java クラスを %IM_HOME%/doc/imart/WEB-INF/classes ディレクトリにコピーしてください。

6.4.3.2 WebLogic でのデプロイ方法

上記で作成した aar ファイルを %IM_HOME%/doc/imart/WEB-INF/services/services.list に追加してください。

なお、WebLogicでは、「6.4.3 aarファイル形式のデプロイ方法」にのみ対応しています。

6.4.3.3 Service Archive Wizardに関する注意

- Axis2 プロジェクトで公開しているEclipseのプラグイン「[Service Archive Wizard](#)」は正しく動作しないのでご注意ください。(2008 年 7 月 7 日現在)

6.5 Axis2 モジュールの適用方法

ある Web サービスを intra-mart の認証・認可の対象とするには、Axis2 モジュール「im_ws_auth」を適用する必要があります。

Axis2 モジュールを Web サービスに適用するには、services.xml や Axis2 の管理コンソールから行うことができますが、ここでは、axis2.xml で設定する方法を説明します。(services.xml の設定方法は「6.4.1 services.xml について」を参照してください)

%IM_HOME%/doc/imart/WEB-INF/conf/axis2.xml の 46 行目付近を編集します。

例えば、Axis2 モジュール「im_ws_auth」を Web サービス「ExampleWebServiceName」に適用するには、以下のように設定してください。(＜module＞、および、＜service＞タグは複数設定することが可能です)

```
<listener class="jp.co.intra_mart.foundation.web_service.axis2.observers.EngageModuleAxisObserver">
  <parameter name="engageModule">
    <module ref="im_ws_auth">
      <service>ExampleWebServiceName</service>
    </module>
  </parameter>
</listener>
```

6.6 認証・認可を必要としない Web サービスの作成方法

intra-mart の認証・認可、および、ログインセッションを必要としない Web サービスを作成する場合、Axis2 モジュール「im_ws_auth」を適用する必要はありません。具体的には、services.xml の＜module ref="im_ws_auth"/＞を設定しない、または、Ant タスク「AntAarGenerateTask」の「moduleRef」属性を設定しないことで、該当モジュールが適用されなくなります。

Axis2 モジュール「im_ws_auth」を適用しないこと以外は、本書で説明している Web サービスと同じ方法で作成することが可能です。

6.7 SOAPフォルトの送信方法

スクリプト開発モデルで Web サービス・プロバイダを構築している場合の SOAP フォルトの送信方法を説明します。なお、JavaEE 開発モデルの場合は、「org.apache.axis2.AxisFault」を利用してください。「AxisFault」の利用方法は Axis2 の API リストを参照してください。

SOAP フォルトの送信には「SOAPFault オブジェクト」を利用します。

以下に JavaScript のサンプルコードを示します。

```
1: // SOAPFault オブジェクトの生成（引数には「Fault Reason」を指定します）
2: var soapFault = new SOAPFault("エラーが発生しました。");
3:
4: // SOAP フォルトの内容を設定します。
5: soapFault.faultCode = "SampleFaultCode";
6: soapFault.faultCodeNamespaceURI = "http://sample.fault/xsd";
7:
8: soapFault.detail = "エラーの詳細情報です";
9: soapFault.detailNodeName = "soapFault_detailNodeName";
10:
11: // SOAPFault をスロー（ここで処理が終了します）
12: soapFault.throwFault();
```

まず、2 行目で、SOAPFault オブジェクトのインスタンスを生成しています。引数には SOAP フォルトの Fault Reason を指定します。4～9 行目にかけて、SOAP フォルトの具体的な内容を設定しています。SOAPFault オブジェクトをスローするには、12 行目のように「throwFault()」関数を実行してください。（「throw soapFault;」といった、JavaScript のスロー構文を利用しないでください）「throwFault()」関数内部で変換処理が行われ、Web サービス・クライアントに SOAP フォルトメッセージが返却されます。

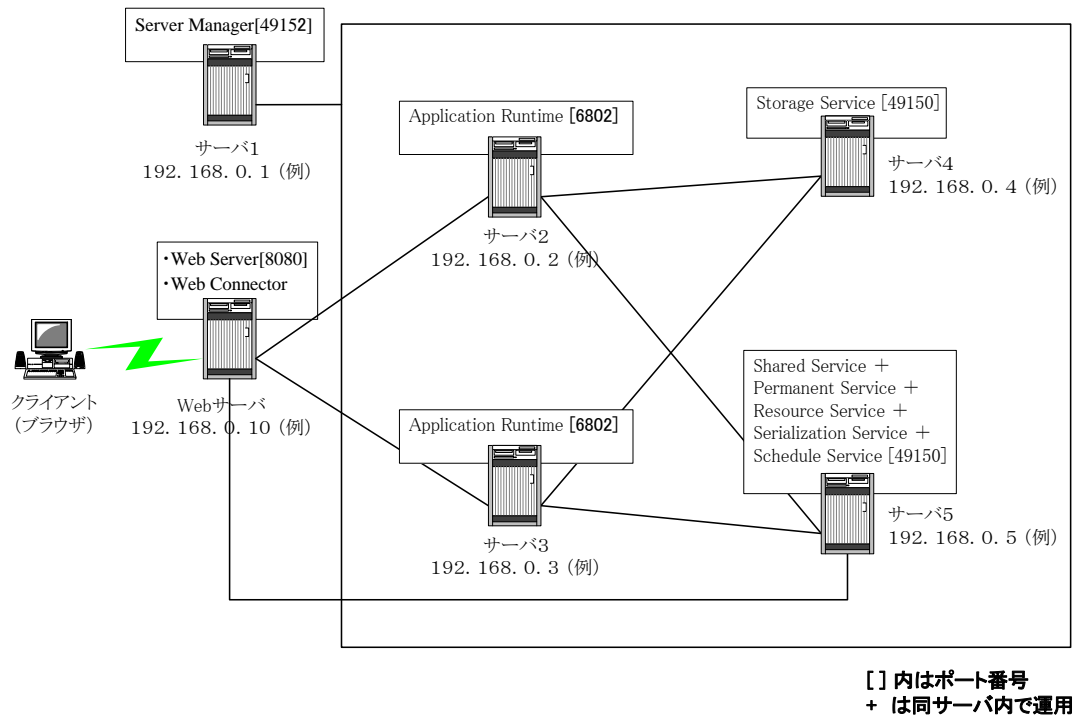
上記サンプルコードで返却される SOAP メッセージの例です。

```
<?xml version='1.0' encoding='UTF-8' ?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
    <wsa:Action>http://www.w3.org/2005/08/addressing/soap/fault</wsa:Action>
    <wsa:RelatesTo>urn:uuid:BBE36157BF34C817391214111551449</wsa:RelatesTo>
  </soapenv:Header>
  <soapenv:Body>
    <soapenv:Fault xmlns:axis2ns4="http://sample.fault/xsd">
      <soapenv:Code>
        <soapenv:Value>axis2ns4:SampleFaultCode</soapenv:Value>
      </soapenv:Code>
      <soapenv:Reason>
        <soapenv:Text xml:lang="en-US">エラーが発生しました。</soapenv:Text>
      </soapenv:Reason>
      <soapenv:Detail>
        <soapFault_detailNodeName>エラーの詳細情報です</soapFault_detailNodeName>
      </soapenv:Detail>
    </soapenv:Fault>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAPFault オブジェクトの詳細は、API リストを参照してください。なお、<soapenv:Text>の「xml:lang」属性は、「en-US」が設定されます。（これは、Axis2 の現行仕様による制限です）

6.8 Application Runtimeを分散させた場合のWSDLについて

Axis2 が自動生成する WSDL に記述されているエンドポイントは、Application Runtime が稼動しているホスト名に依存します。例えば、以下のような構成で intra-mart が構築されていたとします。



WSDL を自動生成する場合、Web サービス「SampleMemberInfoOperatorService」に関するエンドポイントは以下のようになります。

■ サーバ2

```
<?xml version="1.0"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:service name="SampleMemberInfoOperatorService">
    <wsdl:port
      name="SampleMemberInfoOperatorServiceHttpSoap11Endpoint"
      binding="ns:SampleMemberInfoOperatorServiceSoap11Binding">
      <soap:address
        location="http://192.168.0.2:8080/imart/services/
          SampleMemberInfoOperatorService.SampleMemberInfoOperatorServiceHttpSoap11Endpoint"/>
      </wsdl:port>
    <wsdl:port
      name="SampleMemberInfoOperatorServiceHttpSoap12Endpoint"
      binding="ns:SampleMemberInfoOperatorServiceSoap12Binding">
      <soap12:address
        location="http://192.168.0.2:8080/imart/services/
          SampleMemberInfoOperatorService.SampleMemberInfoOperatorServiceHttpSoap12Endpoint"/>
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

■ サーバ3

```

<wsdl:service name="SampleMemberInfoOperatorService">
  <wsdl:port
    name="SampleMemberInfoOperatorServiceHttpSoap11Endpoint"
    binding="ns:SampleMemberInfoOperatorServiceSoap11Binding">
    <soap:address
      location="http://192.168.0.3:8080/imart/services/
        SampleMemberInfoOperatorService.SampleMemberInfoOperatorServiceHttpSoap11Endpoint"/>
    </wsdl:port>
  <wsdl:port
    name="SampleMemberInfoOperatorServiceHttpSoap12Endpoint"
    binding="ns:SampleMemberInfoOperatorServiceSoap12Binding">
    <soap12:address
      location="http://192.168.0.3:8080/imart/services/
        SampleMemberInfoOperatorService.SampleMemberInfoOperatorServiceHttpSoap12Endpoint"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

このように WSDL 内に記述されているエンドポイントの URL が、Web サーバ「192.168.0.10」を経由しない URL のため、Web サービス・クライアントは、この WSDL をそのまま利用して Web サービスを呼び出すことができません。

以降、これを回避するための方法を示します。

6.8.1.1 独自のWSDLを作成する

Webサービスをディレクトリ形式でデプロイした場合(参照:「6.4.2 ディレクトリ形式のデプロイ方法」)、以下のディレクトリに「Webサービス名.wSDL」という名前でWSDLファイルを作成します。

```
%IM_HOME%/doc/imart/WEB-INF/services/【サービス・グループ名】/META-INF/
```

例えば、先ほどの Web サービス「SampleMemberInfoOperatorService」の WSDL を、Web サーバ「192.168.0.10」を経由するようなエンドポイントを書き換えます。

```

<wsdl:service name="SampleMemberInfoOperatorService">
  <wsdl:port
    name="SampleMemberInfoOperatorServiceHttpSoap11Endpoint"
    binding="ns:SampleMemberInfoOperatorServiceSoap11Binding">
    <soap:address
      location="http://192.168.0.10:8080/imart/services/
        SampleMemberInfoOperatorService.SampleMemberInfoOperatorServiceHttpSoap11Endpoint"/>
    </wsdl:port>
  <wsdl:port
    name="SampleMemberInfoOperatorServiceHttpSoap12Endpoint"
    binding="ns:SampleMemberInfoOperatorServiceSoap12Binding">
    <soap12:address
      location="http://192.168.0.10:8080/imart/services/
        SampleMemberInfoOperatorService.SampleMemberInfoOperatorServiceHttpSoap12Endpoint"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

書き換えたWSDLファイルを「SampleMemberInfoOperatorService.wsdl」として、サーバ2 と サーバ3にコピーします。こうすることで、「<http://192.168.0.10:8080/imart/services/SampleMemberInfoOperatorService?wsdl>」で参照可能なWSDLファイルのエンドポイントが、「192.168.0.10:8080」で統一されます。

これにより、Web サービス・クライアントは、この WSDL をそのまま利用して Web サービスを呼び出すことが可能になります。また、Web サーバコネクタを利用して Web サービスの負荷分散が行われるようになります。

6.8.1.1.1 Axis2 ツール「java2WSDL」コマンド

Axis2 プロジェクトが提供している WSDL 生成コマンド「java2WSDL」コマンドがあります。このコマンドは、Web サービスとして公開する Java クラスを指定すると、その Java クラスを解析し、自動的に WSDL を生成します。このコマンドが生成した WSDL のエンドポイントを変更し、各 Application Runtime 上に WSDL ファイルを配備することも可能です。

このコマンドは、Axis2 のバイナリーディストリビューションに付属しています。「java2WSDL」コマンドの詳細は、Axis2 プロジェクトの「[Axis2 Reference Guide](http://ws.apache.org/axis2/1_4_1/reference.html) (http://ws.apache.org/axis2/1_4_1/reference.html)」ページの Java2WSDL Referenceを参照してください。

なお、%AXIS2_HOME%/bin/java2wsdl.bat はそのままでは利用できません。以下のように、Web サービスとして公開する Java クラスの依存ライブラリをクラスパスに追加してから利用することをお勧めします。(依存ライブラリは適宜変更してください)

■ java2wsdl.bat の修正例 (太字部分が追記した箇所です)

```

:okHome
rem set the classes
setlocal EnableDelayedExpansion
rem append the existing classpath to AXIS2_CLASS_PATH
set AXIS2_CLASS_PATH=%CLASSPATH%;%AXIS2_HOME%
rem loop through the libs and add them to the class path
FOR %%c in ("%AXIS2_HOME%\lib\*.jar") DO set AXIS2_CLASS_PATH=!AXIS2_CLASS_PATH!;%%c

rem ----- Execute The Requested Command -----
echo Using AXIS2_HOME: %AXIS2_HOME%
echo Using JAVA_HOME: %JAVA_HOME%
set _RUNJAVA="%JAVA_HOME%\bin\java"

set AXIS2_CLASS_PATH=%AXIS2_CLASS_PATH%;"%WEB_INF%\lib\im_javaee_framework.jar"
set AXIS2_CLASS_PATH=%AXIS2_CLASS_PATH%;"%WEB_INF%\lib\im_javaee_assist.jar"
set AXIS2_CLASS_PATH=%AXIS2_CLASS_PATH%;"%WEB_INF%\lib\im_ws_auth.jar"

%_RUNJAVA% %JAVA_OPTS% -cp "!AXIS2_CLASS_PATH!" org.apache.ws.java2wsdl.Java2WSDL %*
endlocal
:end

```

■ コマンド実行例 (WEB-INF/classes/sample/web_service/provider/ MemberInfoOperatorService.class の例)

```

> set AXIS2_HOME=C:\axis2-1.4
> set IM_HOME=C:\imart
> set WEB_INF=%IM_HOME%\doc\imart\WEB-INF

> cd %AXIS2_HOME%\bin
> java2wsdl.bat -cp %IM_HOME%\doc\imart\WEB-INF\classes                (← 実際は一行です)
-o %IM_HOME%\doc\imart\WEB-INF\services\im_ws_auth_sample\META-INF
-sn MemberInfoOperatorService
-cn sample.web_service.provider.MemberInfoOperatorService

```

「%IM_HOME%\doc\imart\WEB-INF\services\im_ws_auth_sample\META-INF」ディレクトリに、「MemberInfoOperatorService.wsdl」が生成されます。

6.8.1.2 Webサービス・クライアントでエンドポイントを変更する

Web サービス・クライアント側で、Web サーバ「192.168.0.10」を経由するようにエンドポイントを変更します。

スクリプト開発モデル「SOAPClient オブジェクト」を利用して Web サービスを呼び出している場合のエンドポイント変更方法を以下に示します。

「4.2 Webサービス・クライアントの作成」のサンプルを修正します。

SOAPClient オブジェクトのコンストラクタの第 4 引数に Web サービスのエンドポイントを指定します。これを Web サーバ「192.168.0.10」経由の URL に変更します。(下線が変更した箇所です)

```
1:  var wsdlFileURL = "http://192.168.0.10:8080/imart/services/SampleMemberInfoOperatorService?wsdl";
2:  var endPoint    = "http://192.168.0.10:8080/imart/services/SampleMemberInfoOperatorService";
3:  .
4:  .
5:  .
15: /**
16:  * メンバー情報を追加します。
17:  */
18: function add() {
19:
20:     //*****
21:     // ステップ 1 : WSDLを指定して SOAPClientオブジェクト のインスタンスを生成
22:     //*****
23:     try {
24:         var soapClient = new SOAPClient(wsdlFileURL, null, null, endPoint);
25:         Debug.print("ステップ 1 完了");
26:     }
27:     catch(ex) {
28:         Debug.browse("エラーが発生しました。", ex);
29:     }
```


7 制限事項

7.1 全般

1. Webサービスエンジンとして「Axis 2 ver1.4.1」を利用することを前提とします。
Axis2の詳細は [Axis2のWebサイト](#) を参照してください。
2. Web サービスのトランスポート層は HTTP を利用すること前提とします。
3. Web サービスの暗号化は SSL にて実現することを前提とします。
4. document-literal スタイルの Web サービスを公開することを前提とします。
5. Webサービスの負荷分散は、Webサーバコネクタのラウンドロビン機能で実現します。その際のWSDLに関する設定は「6.8 Application Runtimeを分散させた場合のWSDLについて」を参照してください。
6. Application Runtime を分散させた場合、それぞれの Application Runtime に同じ Web サービスをデプロイする必要があります。Application Runtime 間でデプロイされている Web サービスが異なる場合、Web サービスの負荷分散は正しく動作しません。

7.2 Axis2 - 1.4.xの現行仕様

7. Webサービスのメッセージ要素に、継承関係を持つクラスを指定することはできません。詳しくは、「4.1.2.3.1 継承関係を持つ型情報クラスの制限事項」を参照してください。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という、ADB (Axis Data Binding) の現行仕様による制限です。回避するには、ADB以外のデータバインディング方式をご利用ください。
8. Web サービスのメッセージ要素に、内部クラスを指定することはできません。スタブが正しく処理できません。
9. Web サービスとして公開するメソッドの返却値が void で、かつ、例外をスローしない場合、何らかのエラーが発生しても、Web サービス・クライアント側にエラーは通知されません。
10. SOAPMonitor は、Application Runtime と同一マシンで Web サーバが稼動している場合のみ利用可能です。
11. Axis2の管理コンソール (<http://hostname/imart/axis2-admin/index.jsp>) は、URL rewriteing を利用したセッション ID の維持に対応していません。
12. メッセージレシーバ「RPCxxxxMessageReceiver」を利用した XML から Java オブジェクトへのマッピングに関して、「配列」として定義されている Java オブジェクトに対する SOAP メッセージの該当要素がなかった場合、「空の配列」に変換される場合と「null」に変換される場合があります。
13. データバインディング方式に「ADB」を利用した Axis2 のスタブに関して、XML から Java オブジェクトへマッピングされる際、「配列」として定義されている Java オブジェクトが以下のように変換されます。

受信した SOAP メッセージ(XML)	Axis2 スタブでの Java オブジェクト
<ax1:nullArray xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true" />	配列長が 1 の配列 (最初の要素は「null」)
(要素なし)	null

(ア) この現行仕様により、Web サービス・プロバイダ側で「null」を送信した場合、Web サービス・クライアント側では「配列長が 1 の配列 (最初の要素は「null」)」として解釈されたり、Web サービス・プロバイダ側で「空の配列」を送信した場合、Web サービス・クライアント側では「null」として解釈されたりします。

14. Axis2-1.4.1 でAXIS2-3870 が改修されました。しかし、Axis2 のスタブを使用した場合にメモリリークが発生することが確認されました。

[Axis2 1.4.1 client stub not freed](https://issues.apache.org/jira/browse/AXIS2-4007) (<https://issues.apache.org/jira/browse/AXIS2-4007>)

intra-mart WebPlatform/AppFramework ver7.0 パッチ 3 より、AXIS2-4007 を改修したAxis2-1.4.1 を利用しています。改修を行ったクラスは「org.apache.axis2.client.Stub」です。ソースファイルは、本製品 CD-ROM、または、[intra-mart製品最新情報ダウンロードページ](#)の公開ソースファイル「im_ws_auth_client-src.zip」に含まれています。

Axis2 のorg.apache.axis2.client.ServiceClientクラスを利用した場合にメモリリークが発生します。この影響により、スクリプト開発モデルの「SOAPClient」オブジェクト利用時にもメモリリークが発生します。詳しくはApache Software Foundation のJiraを参照してください。

- [Memory Leak using ServiceClient](https://issues.apache.org/jira/browse/AXIS2-3870) (<https://issues.apache.org/jira/browse/AXIS2-3870>)

15. Web サービスとして公開するメソッドの引数に JavaBean が指定されている場合、その JavaBean 内の「バイト配列(=byte[])」形式のプロパティは、データが正しく送受信されません。これは、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メソッドに「バイト配列(=byte[])」形式の引数を指定してください。
16. Webサービスとして公開するメソッドの返却型をvoidとすることはできません。
(<http://issues.apache.org/jira/browse/AXIS2-4275>)

7.3 Webサービスに対する認証・認可

17. intra-mart ログインセッションは、Web サービスの実行後、明示的に破棄(ログアウト)されます。したがって、intra-mart ログインセッションのスコープは Web サービスの呼び出し単位となります。
18. 認証時に利用するパスワードは、常に intra-mart のアカウント情報が保持しているものを利用します。LDAP 連携を行っている場合、LDAP 側で管理されているパスワードは利用されません。
19. intra-mart ユーザの認証・認可は、WS-Security に対応していません。
20. Application Runtime を分散させた場合、Application Runtime 間でデプロイされている Web サービスが異なる場合、ログイングループ管理者の「Web サービスアクセス設定」メンテナンス画面を含む認可機能全般が正しく動作しません。
21. Web サービスのアンデプロイ時に、該当 Web サービスに設定されていたアクセス権限は削除されます。

7.4 Webサービス・プロバイダ

22. 既存の Java クラス、または、JavaScript 関数を Web サービス化することを対象とします。(ボトムアップアプローチによる Web サービス化) WSDL ファイルを作成後、それに則った Web サービスの実装を作成するようなトップダウンアプローチによる Web サービス化は対象外とします。
23. スクリプト開発モデルの Web サービス化に関して、リクエストパラメータを引数とする関数は、Web サービス化の対象外です。リクエストパラメータを引数とする関数とは、init()関数、close()関数、および、リンクやフォームの action 属性に対応する関数を意味します。
24. スクリプト開発モデルの Web サービス化に関して、引数に対して値を更新する処理を行う関数は、Web サービス化の対象外です。引数に対して値を更新する処理を行う関数とは、引数の利用用途が、関数内部で読み取られるだけでなく、関数実行終了後の結果としても利用される関数を意味します。
25. スクリプト開発モデルの Web サービス化に関して、以下の API を利用している関数は、Web サービス化の対象外です。
 - forward()
 - redirect()
 - secureRedirect()
 - transmission()
 - Debug.browse()
 - HTTPResponse.sendMessageBody()
 - HTTPResponse.sendMessageBodyString()
 - Module.download.send()
 - Module.alert.back()
 - Module.alert.link()
 - Module.alert.reload()
 - Module.alert.write()
 - Module.mobile.alert()
26. JavaEE フレームワークのイベントフレームワークの Web サービス化に関して、aar ファイル内に Web サービスとして公開する Java クラスが格納されていると、Web サービスが動作しません。
27. WebSphereでは、独自に作成したWSDL(6.8.1.1参照)を利用するためには、「クラス・ローダーの順序」を「最初にアプリケーション・クラス・ローダーをロードしたクラス」に設定する必要があります。詳しくは、「intra-mart AppFramework Ver.7.0 セットアップガイド」を参照してください。
28. WebLogic では、「6.4.3 aar ファイル形式のデプロイ方法」にのみ対応しています。また、aar ファイル作成後、%IM_HOME%/doc/imart/WEB-INF/services/services.list を編集する必要があります。

7.5 Webサービス・クライアント

29. SOAPClient オブジェクトは、Axis2 の CodeGenerationEngine クラスを利用しています。したがって、CodeGenerationEngine が対応していない Web サービスを呼び出すことはできません。
30. SOAPClient オブジェクトは、データバインディング方式に「ADB」を利用した Axis2 のスタブを使用しています。
31. SOAPClient オブジェクトの getSampleCode()関数は、XML スキーマの restriction で定義されている型などは、サンプルデータが生成されません。サンプルデータが生成されていない型については、WSDL、および、実行する Web サービスの仕様を確認してください。
32. SOAPClient オブジェクトの getSampleCode()関数は、Web サービス・オペレーションの入力メッセージ要素の子要素が「maxOccurs="unbounded"」指定されている場合、正しく表示できません。
33. SOAPClient オブジェクトは、document-literal スタイルの Web サービスに対応しています。
34. SOAPClient は、非同期型コールバック形式の Web サービスに対応していません。
35. SOAPClient オブジェクトを WebSphere で利用するためには、「クラス・ローダーの順序」を「最初にアプリケーション・クラス・ローダーをロードしたクラス」に設定する必要があります。詳しくは、「intra-mart AppFramework Ver.7.0 セットアップガイド」を参照してください。
36. SOAPClient オブジェクトをWebLogicで利用するには、環境変数「AXIS2_HOME」が設定されている必要があります。詳しくは「4.2.4 WebLogicでSOAPClientオブジェクトを利用する方法」を参照してください。
37. SOAPClient オブジェクトは、JavaScript の予約語と同一名称の Web サービス・オペレーションを利用できません。
38. スクリプト開発モデルの SoapClient が返却するオブジェクトのメンバーは、Axis2 が自動生成した JavaBean から PropertyDescriptor を使用してプロパティ名を取得し、同様のプロパティを持つ JS オブジェクトを生成しています。
PropertyDescriptor から返却されるプロパティ名は JavaBean 仕様に基づいたものであるため、SOAP メッセージとは異なるプロパティ名になる場合があります。

例として、SOAP メッセージの要素名が「A0001」である場合、Axis2 はアクセッサメソッドが「getA0001」「setA0001」の JavaBeans を生成します。この JavaBeans からプロパティ名を取得すると「a0001」が返されるため返却値オブジェクトのプロパティ名は「a0001」となります。これは、先頭に一文字だけ存在する大文字英字が JavaBean 仕様のプロパティ名に従い小文字にされるためです。

intra-mart WebPlatform / AppFramework Ver. 7.0
Web サービス・プログラミングガイド

2012/03/26 第 8 版

Copyright 2000-2012 株式会社 NTT データ イントラマート
All rights Reserved.

TEL: 03-5549-2821

FAX: 03-5549-2816

E-MAIL: info@intra-mart.jp

URL: <http://www.intra-mart.jp/>