intra-mart WebPlatform/AppFramework Ver.7.0

スクリプト開発モデル プログラミングガイド

2013/10/18 第 8 版

<< 変更履歴 >>

変更年月日	変更内容
2008/07/07	初版
2008/09/30	第 2 版 「6.4 スクリプト開発モデルの実行処理シーケンスについて」を追記しました。
2009/02/27	第 3 版 「3.11.3 実行時のファンクション・コンテナ検索手順」の説明を修正しました。
2009/06/30	第 4 版 「3.4.2.2 Procedure.define 関数」の誤字を修正しました。
2009/10/30	第 5 版 「3.15 ショートカットアクセス機能」の有効期限の日付指定のサンプルを追加しました。 「3.11.5 コンパイラとは直接関係ない部分の仕様」を修正しました。
2010/11/30	第 6 版 「3.4.3.1 Imart.defineType 関数」を修正しました。
2011/06/30	第 7 版 「3.11.1 自動コンパイル」の記述を修正しました。
2013/10/18	第 8 版 「3.19 ツリー表示」の記述を修正しました。

<< 目次 >>

1 イントロダクション 1
1.1 スクリプト開発モデルでの開発概要 1
1.1.1 プレゼンテーション・ページ 1
1.1.2 ファンクション・コンテナ 2
1.2 各ファイルの保存場所 3
2 スクリプト開発モデルのプログラミング 5
2.1 Hello Worldを作ろう 5
2.1.1 ベースとなるプレゼンテーション・ページ（.html）の作成 5
2.1.2 ファンクション・コンテナ（.js）の作成 7
2.1.3 アプリケーション・プログラムの実行 7
2.2 ページ間にまたがるデータの共有 8
2.2.1 ベースとなるプレゼンテーション・ページ（.html）の作成 8
2.2.2 ファンクション・コンテナ（.js）の作成 9
2.2.3 アプリケーション・プログラムの実行 10
2.3 データベースからデータを取得する 13
2.3.1 ベースとなるプレゼンテーション・ページ（.html）の作成 13
2.3.2 ファンクション・コンテナ（.js）の作成 14
2.3.3 アプリケーションの実行 15
2.4 取得したデータの一覧表示 16
2.4.1 ベースとなるプレゼンテーション・ページ（.html）への追加 16
2.4.2 ファンクション・コンテナ（.js）の作成 16
2.4.3 アプリケーションの実行 17
2.5 データの登録・更新・削除 19
2.5.1 ベースとなるプレゼンテーション・ページ（.html）の作成 19
2.5.2 ファンクション・コンテナ（.js）の作成 20
2.5.3 アプリケーションの実行 21
3 Javaコンポーネント群（im-BizAPI）の利用 23
3.1 画面共通モジュール 23
3.1.1 画面デザイン共通モジュール 23
3.1.2 標準画面の作り方（共通画面デザイン） 25
3.2 Storage Serviceの利用方法 26
3.2.1 ファイル・アップロード 26
3.2.2 ファイルリストの表示 28
3.2.3 ファイル・ダウンロード 30
3.2.4 ファイルの削除 32
3.3 メール連携モジュール（ファンクション・コンテナ） 35
3.3.1 メール送信 35
3.3.2 添付ファイル付きメールの送信 38
3.4 共通ライブラリの作成 41
3.4.1 拡張APIの作成 41
3.4.2 グローバル関数の作成 45
3.4.3 拡張<IMART>タグの作成 48
3.5 外部プロセスの呼び出し 51
3.6 JavaClassとの連携 52
3.6.1 標準JavaClassとの連携方法 52
3.7 EJBとの連携 56

3.7.1 EJBコンポーネントの作成 56
3.7.2 JavaScriptからの呼び出し 56
3.8 XML形式のデータを扱う 57
3.8.1 XMLパーサーとデータの取得 57
3.8.2 XML形式データの受信方法 57
3.8.3 XML形式データの送信方法 59
3.9 E4Xの利用方法 62
3.9.1 E4Xとは？ 62
3.9.2 XMLオブジェクトの作成 62
3.9.3 値の取得 62
3.9.4 子ノードの取得 63
3.9.5 ノードの追加 63
3.9.6 ノードの削除 63
3.9.7 変数の挿入 64
3.10 JSSP-RPCについて 65
3.10.1 動作イメージ 65
3.10.2 JSSP-RPC 通信エラーオブジェクトに関して 65
3.10.3 JSSP-RPC サンプルプログラム（同期通信） 66
3.10.4 JSSP-RPC サンプルプログラム（非同期通信） 67
3.11 JavaScriptコンパイラ機能について 69
3.11.1 自動コンパイル 69
3.11.2 手動コンパイル 72
3.11.3 実行時のファンクション・コンテナ検索手順 75
3.11.4 仕様詳細 75
3.11.5 コンパイラとは直接関係ない部分の仕様 76
3.11.6 制約 76
3.12 im-JavaEE Frameworkとの連携 79
3.13 グラフ描画モジュール 80
3.14 アクセスコントローラモジュール 82
3.15 ショートカットアクセス機能 83
3.16 外部ライブラリコール 85
3.16.1 概要 85
3.16.2 外部ライブラリコールのアーキテクチャ 85
3.16.3 Javaラッパークラスと外部ライブラリの作成 85
3.16.4 外部ライブラリコールの方法 88
3.17 バッチ管理モジュール 89
3.17.1 プログラムの作成 89
3.17.2 システム構成 89
3.17.3 バッチの登録と設定 89
3.18 カレンダーunit 90
3.18.1 呼び出し方法 90
3.18.2 カレンダーデータの受け取り方法 90
3.18.3 カレンダー拡張タグ と カレンダーモジュール 91
3.19 ツリー表示unit 94
3.20 アプリケーション・ロック機能 96
3.21 一意情報の取得機能 98
3.22 製品のカスタマイズ 99

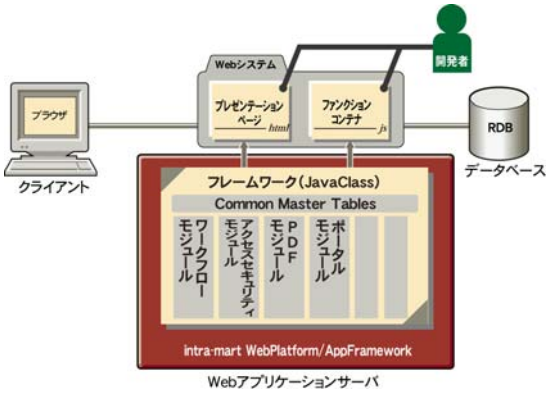
Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.22.1 規定 99
3.22.2 環境移行の手順 99
3.22.3 注意事項 99
3.23 アクセスセキュリティモジュールを利用しないで画面を構築する方法 100
3.23.1 概要 100
3.23.2 準備(インストール) 100
3.23.3 作成(スクリプト開発モデル) 100
3.23.4 注意事項 101
3.24 検索ストリーミング機能 102
3.25 データベースのストアドプロシージャの呼び出し 103
3.26 国際化対応 104
4 デバッグ 106
4.1 デバッグ手順 106
4.1.1 デバッグ例 106
4.1.2 デバッグAPIの利用方法 107
4.2 単体テスト環境（im-JsUnit） 110
4.2.1 im-JsUnit概要 110
4.2.2 テストケースの実行順序 111
4.2.3 テストケースの作成 112
4.2.4 テストランチャーの作成 114
4.2.5 単体テストの実行 114
5 サンプルプログラムの実行 115
5.1 サンプルのインストール 115
5.2 メニューのサンプル実行 116
5.3 APIリスト 117
6 Appendix 118
6.1 メッセージ設定 118
6.2 予約語一覧 118
6.3 制限事項 118
6.3.1 ファイル名称 118
6.3.2 ID、コード 118
6.3.3 JavaScript関数 118
6.4 スクリプト開発モデルの実行処理シーケンスについて 119
6.4.1 ファンクションコンテナの種類 120

1 イントロダクション

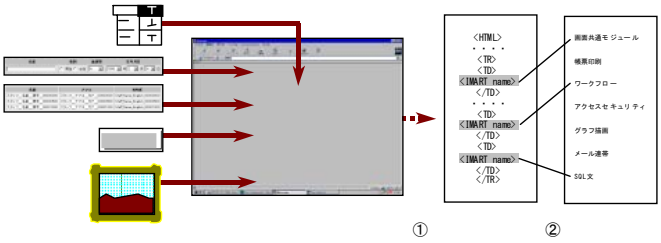
1.1 スクリプト開発モデルでの開発概要

intra-mart WebPlatform/AppFramework を使ったアプリケーションの開発において、開発者はブラウザ上に表示するコンテンツと、サーバ上で動作するビジネスロジックを作成することになります。
スクリプト開発モデルではプレゼンテーション・ページ（HTML ファイル）とファンクション・コンテナ（サーバサイド JavaScript ファイル）の 2 つのファイルを作成します。HTML と Java スクリプトで開発を行えるため、ホームページ作成の延長で、データベースと連動した本格的な Web システムの開発が可能になります。intra-mart WebPlatform/AppFramework に用意されているモジュール群を活用することで、さらに生産性を向上させることができます。



1.1.1 プレゼンテーション・ページ

プレゼンテーション・ページは、ユーザインタフェース部分に相当します。拡張子は「.html」です。開発者は、“intra-mart eBuilder”、または、任意のエディタを利用して、ブラウザに表示するコンテンツを作成していきます。（動画像やサウンドなどを盛り込んだマルチメディアリッチな画面を構築していくことも可能です） ホームページ作成ツールから生成される HTML ファイルに<IMART>タグを追加していくことで、ファンクション・コンテナの実行結果を関連付けることが可能になります。ユーザ定義関数を呼び出す<IMART>拡張タグを追加することもできます。さらに、プレゼンテーション・ページは HTML ファイルであるため、ユーザインタフェース部分のみを切り出してホームページデザイナに作業を依頼するなど、Web システム開発の分業を行うことが可能です。



- 市販のホームページ作成ツールや intra-mart「eBuilder」を利用してユーザインタフェースとなるプレゼンテーション・ページのひな型を作成します。
- ホームページ作成ツールから自動生成された HTML ソースに、ファンクション・コンテナ中の JavaScript 関数や intra-mart WebPlatform のオブジェクトと連携する<IMART>タグを intra-mart

「eBuilder」やテキストエディタを使い挿入（記述）します。

プレゼンテーション・ページのサンプル例を示します。
intra-mart の独自拡張タグ<IMART>を利用して、各種モジュールを呼び出していきます。

＜社員マスタからのデータ取得用 HTML（一覧表示用）ー項目を拡張＞

```
  1: <HTML>
  2: <BODY>
  3: <TABLE border="1">
  4: <TR>
  5:     <TD>社員コード</TD>
  6:     <TD>社員名</TD>
※ 7:     <TD>社員名（カナ）</TD>
  8: </TR>
  9: <IMART type="repeat" list=staffList item="record">
 10: <TR>
 11:     <TD><IMART type="string" value=record.staff_cd></IMART></TD>
 12:     <TD><IMART type="string" value=record.stf_name_kanji></IMART></TD>
※ 13:     <TD><IMART type="string" value=record.stf_name_kana></IMART></TD>
 14: </TR>
 15: </IMART>
 16:  </TABLE>
 17: </BODY>
 18: </HTML>
```

intra-mart の独自拡張タグ<IMART>を利用して、各種モジュールを呼び出していきます。

1.1.2 ファンクション・コンテナ

ファンクション・コンテナは、サーバ上で稼動するビジネスロジック部分に相当します。拡張子は「.js」です。ファンクション・コンテナとプレゼンテーション・ページはワンセットとなっているため、ファイルラベル名は同一のものを使用します。開発者は、intra-mart WebPlatform/AppFramework で用意されているモジュール郡から必要なオブジェクトや関数を選び出し、サーバサイドで稼動するビジネスロジックを Java スクリプトで作成していきます。（“intra-mart eBuilder”を利用することで生産性が向上します）データベースの SQL 文もファンクション・コンテナの中に記述していきます。データベースの接続や SQL 発行は、intra-mart WebPlatform/AppFramework のモジュールが行うため、開発者は、細かなセッション管理やトランザクション管理を意識する必要はありません。ビジネスロジックの実行結果は、プレゼンテーション・ページの<IMART>タグによって関連付けられ、ブラウザ上に表示されます。intra-mart WebPlatform/AppFramework で用意されているモジュール郡の詳細は「API リスト」に記述されています。

このように、スクリプト開発モデルでは、HTML と Java スクリプトで開発を行えるため、ホームページ作成の延長で、データベースと連動した本格的な Web システムの開発が可能になります。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

1.2 各ファイルの保存場所

intra-mart WebPlatform/AppFramework の各ファイルの保存場所を示します。

- intra-mart WebPlatform（Resin）の場合
  - 静的コンテンツ（HTML ファイルや画像ファイルなど）
    Web サーバコネクタのインストールディレクトリ以下
    （スタンドアロン型の場合はサーバモジュールのインストールディレクトリ直下 doc/imart ディレクトリ以下）

  - スクリプト開発モデルのプログラム
    （プレゼンテーション・ページ(.html), ファンクション・コンテナ(.js)）
    ソースディレクトリ以下（通常は、%ResourceService%/pages/src/）

  - Storage Service により一元管理されるファイル群
    %Storage Service%/storage/ディレクトリ内

- **intra-mart WebPlatform（JBoss）および intra-mart AppFramework の場合**
  - 静的コンテンツ（HTML ファイルや画像ファイルなど）
    フレームワークサーバのインストールディレクトリ直下 doc/imart ディレクトリ以下

  - スクリプト開発モデルのプログラム（プレゼンテーション・ページ(.html), ファンクション・コンテナ(.js)）
    ソースディレクトリ以下（通常は、%ResourceService%/pages/src/）

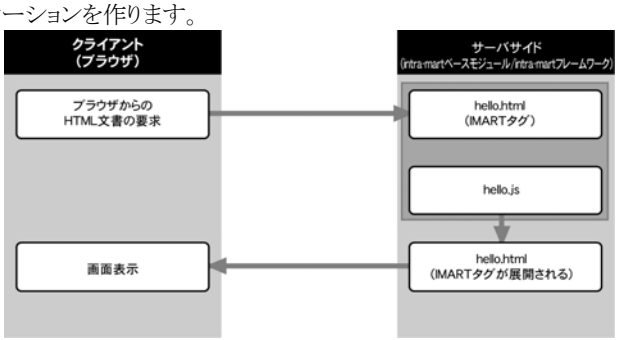
  - Storage Service により一元管理されるファイル群
    %Storage Service%/ storage/ディレクトリ内

2 スクリプト開発モデルのプログラミング

2.1 HelloWorldを作ろう

ここでは、intra-mart のスクリプト開発モデルを用い簡単なアプリケーションを作成する作業を通して、プレゼンテーション・ページやファンクション・コンテナの作成の実際について理解を深めましょう。

ここでは、簡単な例としてブラウザからサーバ上に作成したプレゼンテーション・ページである hello.html を起動させたとき、サーバサイドの intra-mart WebPlatform と連携して「こんにちは、イントラマートです。」とブラウザ上に表示させるアプリケーションを作ります。



2.1.1 ベースとなるプレゼンテーション・ページ（.html）の作成

ブラウザ上に表示させるためのプレゼンテーション・ページを、HTML 形式で作成します。
HTML は、intra-mart の「eBuilder」や市販のホームページ作成ツール、エディタを利用して作成します。ここでは、下記のように画面上に文字列を表示するだけのシンプルな静的 HTML を最初に作成します。ファイル名は hello.html とし、以下の場所へ保存してください。

%ResourceService%/pages/src/sample/user1

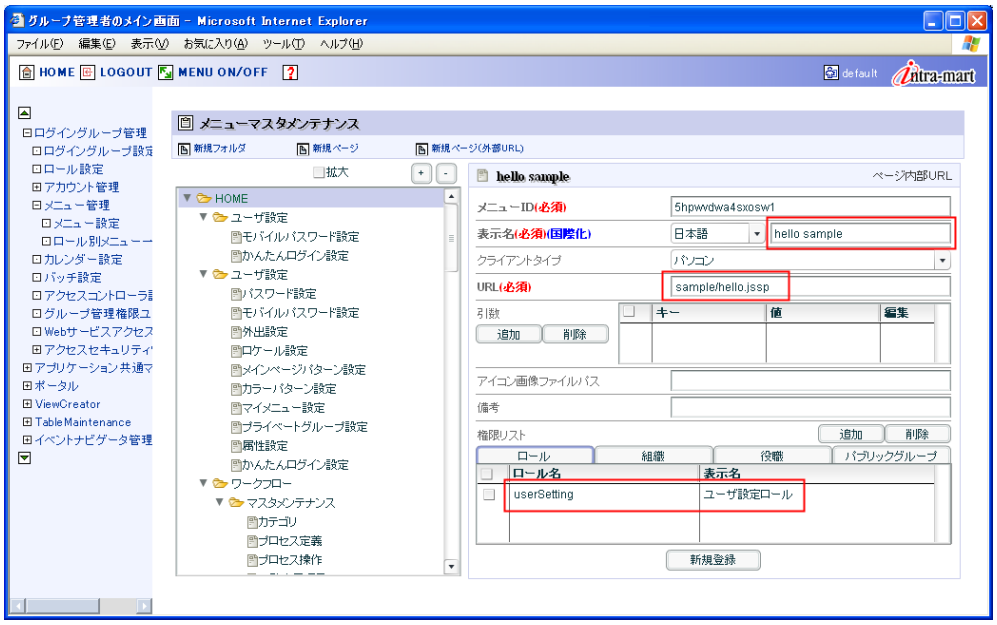
```
<作成した hello.html ファイル >
<HTML>
        <BODY>
        こんにちは、intra-mart です。
  </BODY>
</HTML>
```

作成したファイルを保存したら、次の手順でメニューの登録を行います。（以降はメニュー登録の手順を省略します。）

■ メニューの登録

1. ログイングループ管理者でログインする。
   - URL：http://intramart/imart/(ログイングループ ID).manager
   - ID、パスワードを入力
2. メニューの設定を行う。
   - 「メニュー」－「ログイングループ管理」－「メニュー管理」－「メニュー設定」を選択。
   - 「新規ページ」をクリックし、以下の設定を行う。

```
表示名 ： hello sample
URL ： sample/user1/hello.jssp
「権限リスト」－「追加」－： UserSetting
```



3. 一般ユーザでログインする。

- URL：http://intramart/imart/(ログイングループ ID).portal
- ID、パスワードを入力

4. 「メニュー」－「hello sample」を選択すると、以下の画面が表示されます。

こんにちは、intra-martです。

<hello.html の実行画面>

作成した HTML ファイル上で、ファンクション・コンテナと連携させたい部分に<IMART>タグを埋め込み、プレゼンテーション・ページを完成させます。ここでは、「intra-mart」の部分を<IMART>タグにし、ファンクション・コンテナで指定した文字列（nameValue の値）に置き換えるように設定します。

＜IMART タグを埋め込んで修正した hello.html ファイル＞

```
<HTML>
  <BODY>
        こんにちは、<IMART type="string" value=nameValue></IMART>です。
    </BODY>
</HTML>
```

■ 解説

◆ <IMART type="string"></IMART>

ここでは、<IMART>タグの type 句に「string」を指定しています。「string」は、value 句に指定された変数をファンクション・コンテナ中の値に置き換えるための属性です。 type 句に指定できる属性には他にも「link」、「repeat」、「form」、「input」、「select」など、プレゼンテーション・ページとファンクション・コンテナを連携させるためのものが、intra-mart WebPlatform により多数用意されています。intra-mart で提供している<IMART>タグの詳細は、intra-mart WebPlatform に付属している「API リスト」を参照してください。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

2.1.2 ファンクション・コンテナ（.js）の作成

作成したプレゼンテーション・ページに対応するファンクション・コンテナを作成します。ファンクション・コンテナには、初期化関数である init 関数を作成します。ここでは、文字列「イントラマート」を「nameValue」という名前のプロパティに設定します。hello.html と連携させるため、ファイル名は hello.js とし、プレゼンテーション・ページと同じフォルダに置きます。

<作成したファンクション・コンテナ（hello.js）>

```
// HTML へ渡す値の宣言
var nameValue;

// init 関数の定義
function init(request){
   nameValue = "イントラマート";                // HTML へ渡す値を設定します
}
```

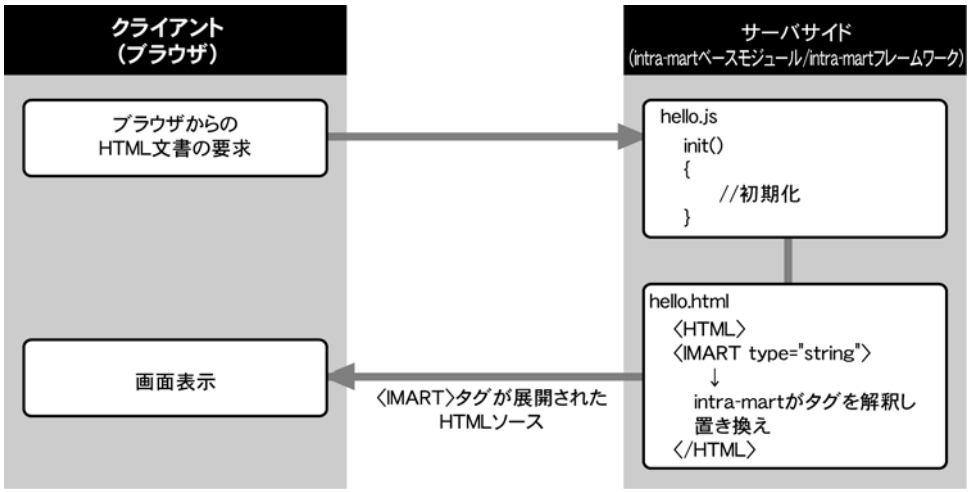
※ ファンクション・コンテナの中の関数 init()は、intra-mart WebPlatform が自動的に初期解釈する固定関数です。

2.1.3 アプリケーション・プログラムの実行

作成したプレゼンテーション・ページとファンクション・コンテナからなるアプリケーションを実行すると、最初に示したように「こんにちは、イントラマートです。」と表示されます。

こんにちは、イントラマートです。

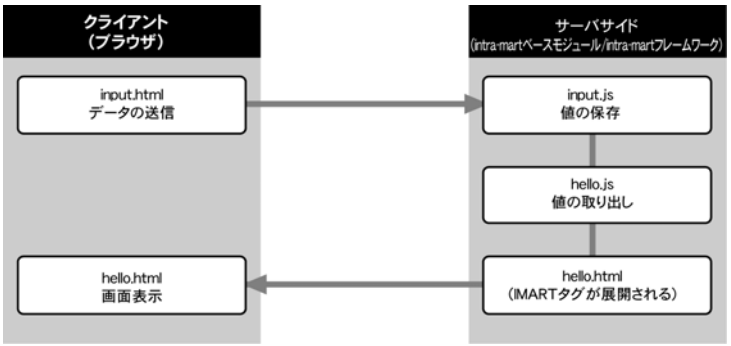
<実行時の結果画面>



※ 任意のディレクトリに作成したアプリケーションを実行するには、ログイングループ管理者でログインし、[ログイングループ管理]-[メニュー管理]-[メニュー設定]でアプリケーションのページ登録をする必要があります。メニューの登録の仕方は「2.1.1 ベースとなるプレゼンテーション・ページ（.html）の作成」を参照してください。

2.2 ページ間にまたがるデータの共有

intra-mart を利用すると、複数ページ間でのデータ共有（セッション管理）が簡単に行えます。
ここでは、プレゼンテーション・ページからデータを入力し、ファンクション・コンテナを通してサーバ側でデータを保存、別のページにデータを表示するというアプリケーションを作ってみます。
ここでは、名前を入力するページ（input.html）を新しく作成して、そこで取得したデータを、前節で作成した hello.html に表示させます。



2.2.1 ベースとなるプレゼンテーション・ページ（.html）の作成

基本となるプレゼンテーション・ページを作成します。拡張子は、.html に限定されています。

2.2.1.1 送信側の HTML ファイルの準備（input.html）

一般的な Web ページと同じように<FORM>タグを使って、データ入力をする HTML ページを最初に作成します。
ファイル名を input.html とし、%ResourceService%/pages/src/sample/user1 に保存します。

<送信フォーム input.html>

```
<HTML>
<BODY>
  名前を入力してください。<BR>
  <FORM method="POST">
      <INPUT type="text" name="yourname"><BR>
      <INPUT type="submit" value="送信">
      </FORM>
</BODY>
</HTML>
```

フォームの送信ボタンが押されたときに、ファンクション・コンテナ上で指定した関数を呼び出すようにして、フォームで入力された値をファンクション・コンテナ側で参照できるようにします。
そのために、ファンクション・コンテナで連携させる必要のある<FORM>タグおよび、<INPUT>タグを<IMART>タグに置き換えます。
完成した input.html ファイルを保存します。

<ファンクション・コンテナとの連携用送信フォーム HTML>

```
<HTML>
<BODY>
名前を入力してください。<BR>
<IMART type="form" method="POST" action="inputName" page="sample/user1/hello">
      <IMART type="input" style="text" name="yourname"></IMART><BR>
      <IMART type="submit" value="送信"></IMART>
</IMART>
</BODY>
</HTML>
```


Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

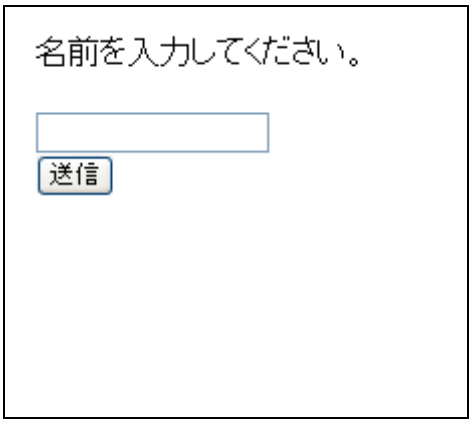
2.2.1.2 表示側の HTML ファイルの準備（hello.html）

作成した HTML ソースは直接ブラウザでも表示できますが、ここでは、前章で作成した hello.html からリンクをして input.html を表示するようにしてみます。
hello.html に次の行を追加してください。

<hello.html に追加>

```
<IMART type="link" page="保存フォルダ名/input">名前の入力</IMART>
```

※ 保存フォルダ名とは、ここでは%ResourceService%/pages/src/sample/user1 を指しています。



<input.html 実行時の画面>

■ 解説

◆ <IMART type="link"> ～ </IMART>
このタグを利用する事で intra-mart のセッションを維持しながらプログラムを動作させることができます。詳細は、「API リスト」を参照してください。

2.2.1.3 目的の HTML ソースへのパス

intra-mart WebPlatform の Resource Service の管理しているソースディレクトリ（ 標準では、%Resource Service%/pages/src/ ）からの相対パスとなります。また、拡張子は指定しません。
"./input" や "保存フォルダ/input.html"という記述をするとエラーになります。

2.2.2 ファンクション・コンテナ（.js）の作成

次に、JavaScript でファンクションコンテナファイルを作成します。

2.2.2.1 受取側の JavaScript ファイルの作成（input.js）

<データ受け取り保存する Java スクリプト>

```
// inputName 関数の定義
function inputName(request) {
  // 受け取った値を Client オブジェクトに保存する
      Client.set( "nameValue", request.yourname );
}
```

2.2.2.2 保存しておいたデータを呼び出す JavaScript ファイルの作成(hello.js)

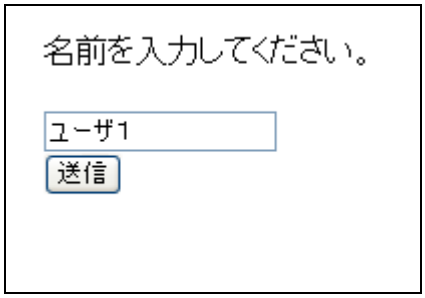
入力した値を参照するように、前節で作成した hello.js の次の記述を下のように書き換えてみてください。

<保存しておいたデータを取り出す例>

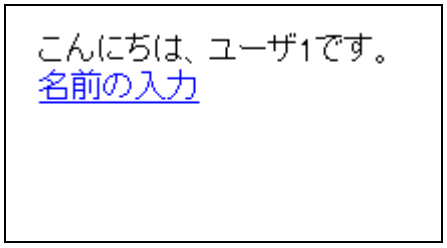
```
nameValue = "イントラマート";            // HTML へ渡す値を設定
                ↓
nameValue = Client.get( "nameValue" );          // HTML へ渡す値を設定
```

2.2.3 アプリケーション・プログラムの実行

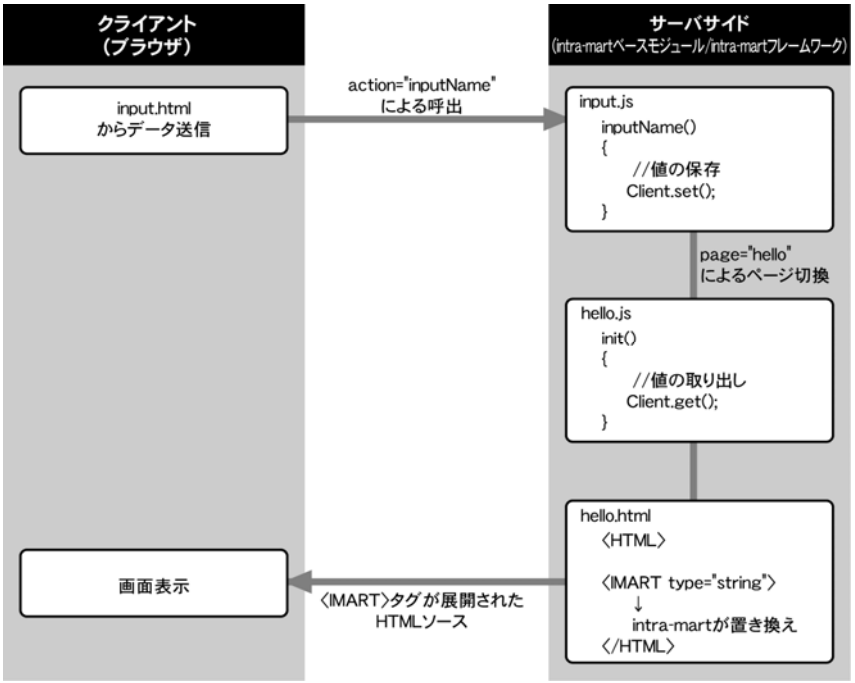
作成したプレゼンテーション・ページとファンクション・コンテナからなるアプリケーションを実行すると、送信側 input.html で入力して送信したデータ「ユーザ 1」が、受信側 hello.html に表示されます。



<送信側 input.html>



<受信側 hello.html>



Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

■ 解説

◆ **<IMART type="form"> ～**

input.html の4行目で使っている<IMART type="form">について、もう少し詳しく説明します。

<IMART type="form" method="POST" action="inputName"page="sample/user1/hello">

この<IMART type="form">は、intra-mart ファンクション・コンテナにデータを引き渡すための<FORM>タグを提供する指定方法です。

action 属性には、フォームのデータがサーバに送信されたときに呼び出される、ユーザ定義関数名を指定します。input.js を見ると、サーバサイドのファンクション・コンテナ上に同じ名前の関数が定義されています。

page 属性は、フォームデータの送信とサーバ側での処理が完了した後に表示したいページを指定できます。省略した場合は現在のページを再描画します。

ここでも、リンク先の指定方法は、Resource Service の管理しているソースディレクトリ（標準では%Resource Service%/pages/src）からの相対パスになりますので注意してください。

◆ **request オブジェクト**

データを受信したファンクション・コンテナでは、request オブジェクトを利用してプレゼンテーション・ページのデータを参照できます。request オブジェクトは、intra-mart WebPlatform/AppFramework から自動的に呼び出される関数（init()や<IMART>タグの action 属性への指定関数）の引数として受け取ることができます。

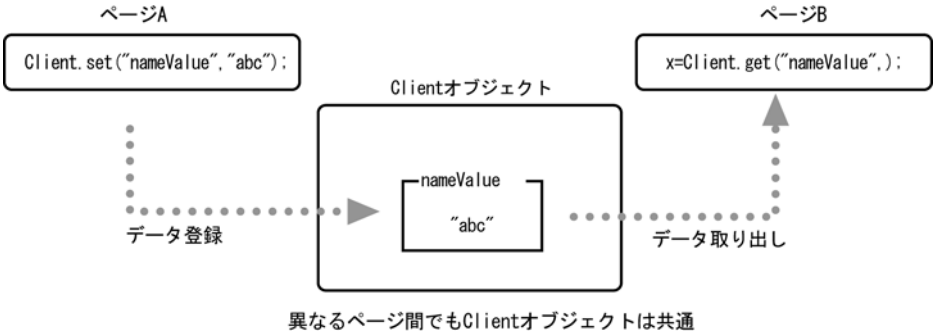
Client.set("nameValue", request.yourname);プレゼンテーション・ページからデータが送られると、input.js の inputName 関数が呼び出されます。ファンクション・コンテナ側では、request オブジェクトのプロパティ名として yourname（input.html では、テキスト入力フィールドに yourname という名前を付けていました）を指定するだけで、簡単に送信されてきたデータを取り出すことができます。

◆ **Client オブジェクトによるセッション管理**

別のページへ移動した後もサーバ側でデータを保存しておくために、Client オブジェクトの set メソッドを使っています。

Client.set メソッドは、クライアントの Web ブラウザが intra-mart で作成したサーバアプリケーションに接続している間、データを保持するように指示するメソッドです。保有するデータには名称を付けることができ、複数保存することも可能です。

この例では、nameValue という名前を付けて保存しています。



intra-mart WebPlatform/AppFramework が提供するオブジェクトには、他にも便利なメソッドやオブジェクトが多数定義されています。アプリケーションの開発者は、これらのメソッドやモジュールを利用することで、短い開発期間で品質の優れたアプリケーションを構築することが可能になります。

各クライアント情報の保持時間の制限（セッションタイムアウト値）

intra-mart では、ここで述べたセッション管理情報やアクセスセキュリティ情報など各クライアントごとの情報を一定時間 Application Runtime 上のメモリ（HttpSession）に保持しています。

デフォルト時間の設定は 10 分となっており、10 分以上クライアントからのアクセスがない場合には、再度ログインし直す必要があります。このデフォルトの時間設定は、conf/http.xml（基本設定ファイル）で変更することができます。conf/http.xml の編集に関しては、WebPlatform 設定ガイドを参照してください。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

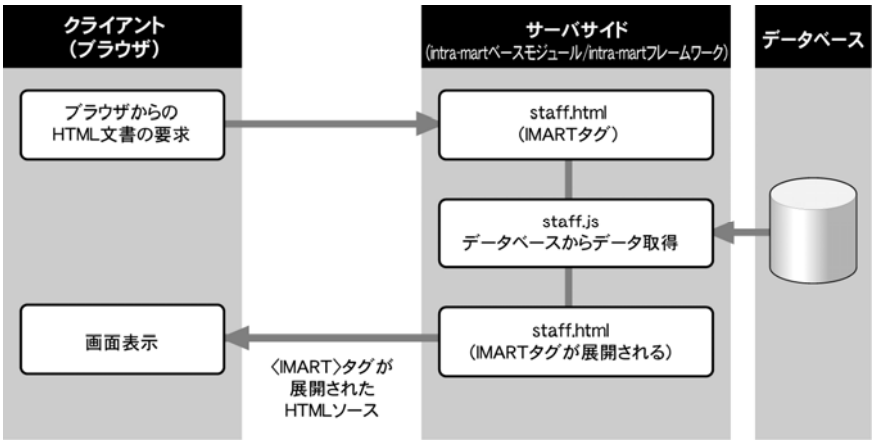
2.3 データベースからデータを取得する

ここでは、intra-mart が用意しているオブジェクトやメソッドの中から、データベースアクセスの為のオブジェクトおよびメソッドを利用して実際のデータベースからデータを取得します。
前章の手順に従って、サーバ側で動的に生成されるページを作成していきます。

まずは下記表のようなデータベースを用意してください。

社員マスタ（m_sample_stf）				
列名	内容	属性	データ型	サイズ（bytes）
staff_cd	社員コード	主キー	テキスト型	20
stf_name_kanji	社員（漢字）		テキスト型	50
stf_name_kana	社員（カナ）		テキスト型	50
stf_name_eng	社員（英字）		テキスト型	50

＜利用するデータベース＞



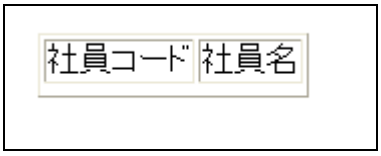
＜データ取得の流れ＞

2.3.1 ベースとなるプレゼンテーション・ページ（.html）の作成

次の HTML ファイルは、社員コードと社員名を表示するための通常の HTML ソースの例です。
ホームページ作成ツールや intra-mart「eBuilder」などを利用すると表の作成が簡単に行えるので、プレゼンテーション・ページの最初のひな型作成には、これらのツールを利用すると便利です。

<社員コードと社員名を表示する staff.html>

```
<HTML>
<BODY>
<TABLE border="1">
<TR>
  <TD>社員コード</TD>
  <TD>社員名</TD>
</TR>
<TR>
  <TD></TD>
  <TD></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```



<staff.html ファイルの実行結果>

ファンクション・コンテナからのデータを反映するために修正した staff.html を次に示します。

<社員マスタからのデータ取得用 staff.html>

```
<HTML>
<BODY>
<TABLE border="1">
<TR>
  <TD>社員コード</TD>
  <TD>社員名</TD>
</TR>
<TR>
  <TD><IMART type="string" value=staff_code></IMART></TD>
  <TD><IMART type="string" value=staff_name></IMART></TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

2.3.2 ファンクション・コンテナ（.js）の作成

ファンクション・コンテナの init 関数を作成します。ファイル名は staff.js とします。

この staff.js では、次の2つのことを実現します。

① データベース上の社員マスタからデータを取得します。
② 取得したデータを HTML へ引き渡します。

<社員マスタからのデータ取得用 staff.js>

```
// HTML へ渡す値を宣言します
  var staff_code;
  var staff_name;

// init 関数の定義
function init(request)
{
  var objData   = false;   // データベースから取得したデータ格納用

  // データベースから社員データを全て取得します
  objData = DatabaseManager.select("SELECT * FROM m_sample_stf");

  if(objData.data.length != 0){
  // HTML へ渡すデータのリストを設定します
      staff_code = objData.data[0].staff_cd;
      staff_name = objData.data[0].stf_name_kanji;
  }else{
      staff_code="";          // データを取得できなかった場合
      staff_name="";
  }
}
```

※ staff_cd、stf_name_kanji はデータベースの列名です。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

■ 解説

◆ **DatabaseManager オブジェクト－1**

データベースへのアクセスは DatabaseManager オブジェクトを通して簡単に行えます。

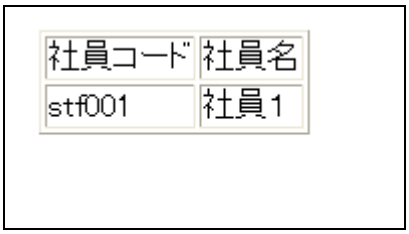
この例では、DatabaseManager オブジェクトの「select」メソッドを呼び出しています。このようにパラメータに SQL 形式の SELECT 文を記述してデータベースからデータを取得することができます。

取得したデータはオブジェクトとして返されます。データベースから取得したレコードは、select メソッドが返すオブジェクトのもつ配列プロパティとして保存されています。名称は「data」です。select メソッドが返すオブジェクトには他にも、取得できたレコード数を保有する「countRow」などがあります。

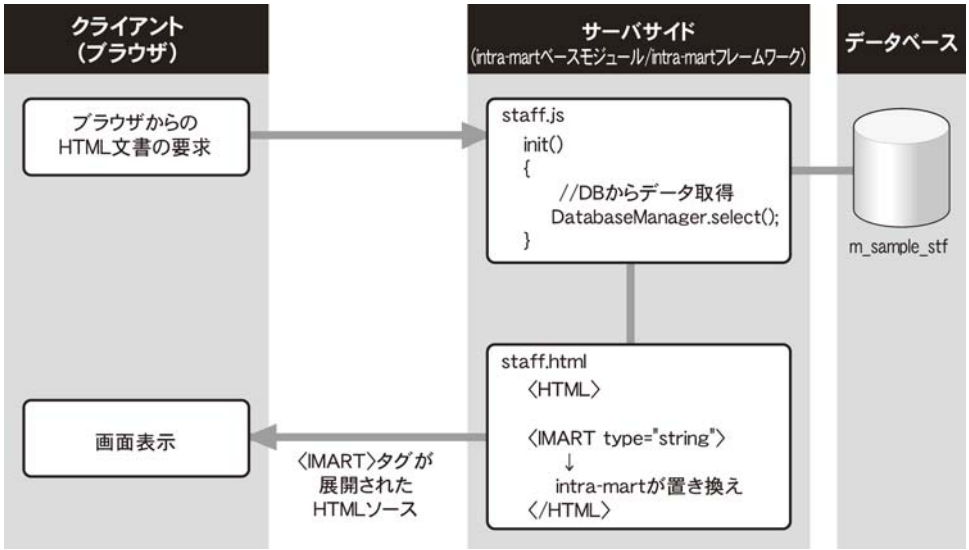
intra-mart WebPlatform が提供するオブジェクトとメソッドの詳細は、intra-mart WebPlatform に付属する「API リスト」を参照してください。

2.3.3 アプリケーションの実行

このアプリケーションを実行した結果を次に示します。



<実行結果>



※ 作成した staff.js ファイルの 10、11 行目の date[0]を date[1]、data[2]に変更してみてください。2 件目、3 件目のレコードが表示されるようになります。

2.4 取得したデータの一覧表示

前節で作成した社員名表示ページ（staff）を改良して、社員名一覧を表示するようにソースを修正します。

2.4.1 ベースとなるプレゼンテーション・ページ（.html）への追加

一覧の表示は、<IMART type="repeat">タグを利用します。repeat は、タグで囲まれた範囲を指定回数だけ繰り返し実行して HTML へ展開します。 さらに、Java スクリプト側で作成した配列と連動して、配列内のデータを順次表示することもできます。

<社員マスタからのデータ取得用 HTML（一覧表示用）>

```
<HTML>
<BODY>
<TABLE border="1">
<TR>
  <TD>社員コード</TD>
  <TD>社員名</TD>
</TR>
<IMART type="repeat" list=staffList   item="record">
<TR>
  <TD><IMART type="string" value=record.staff_cd></IMART></TD>
  <TD><IMART type="string" value=record.stf_name_kanji></IMART></TD>
</TR>
</IMART>
 </TABLE>
</BODY>
</HTML>
```

2.4.2 ファンクション・コンテナ（.js）の作成

次に JavaScript ファイルを示します。

<社員マスタからのデータ取得用 Java スクリプト（一覧表示用）>

```
//HTML へ渡す値を宣言します
var staffList;

// init 関数の定義
function init()
{
      var objData   = false;   //  データベースから取得したデータ格納用

      //  データベースから社員データを全て取得します
            objData = DatabaseManager.select( "SELECT * FROM m_sample_stf" );

            // HTML へ渡すデータのリストを設定します
            staffList = objData.data;

}
```

m_sample_stf は、社員マスタテーブルです。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

2.4.3 アプリケーションの実行

次にアプリケーションを実行した結果を示します。

社員コード	社員名
stf001	社員1
stf002	社員2
stf003	社員3
stf004	社員4
stf005	社員5
stf006	社員6
stf007	社員7
stf008	社員8
stf009	社員9
stf010	社員10

<実行結果>

■ 解説

◆ 項目の拡張

完成したアプリケーションには、「社員名（カナ）」の項目を追加することができます。変更するのはプレゼンテーション・ページ（staff.html）だけで、ファンクション・コンテナは変更の必要はありません。下記のリストでは、←の行を変更しました。

<社員マスタからのデータ取得用 HTML（一覧表示用）－項目を拡張>

```
<HTML>
<BODY>
<TABLE border="1">
<TR>
  <TD>社員コード</TD>
  <TD>社員名</TD>
  <TD>社員名（カナ）</TD>                                    ← 変更した行
</TR>
<IMART type="repeat" list=staffList   item="record">
<TR>
  <TD><IMART type="string" value=record.staff_cd></IMART></TD>
  <TD><IMART type="string" value=record.stf_name_kanji></IMART></TD>
  <TD><IMART type="string" value=record.stf_name_kana></IMART></TD>   ← 変更した行
</TR>
</IMART>
</TABLE>
</BODY>
</HTML>
```

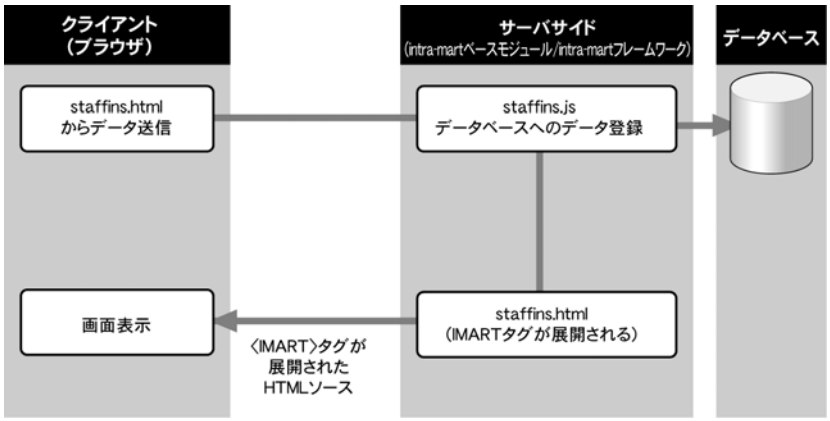
ID	社員名	社員名(カナ)
stf001	社員1	シャイン1
stf002	社員2	シャイン2
stf003	社員3	シャイン3
stf004	社員4	シャイン4
stf005	社員5	シャイン5
stf006	社員6	シャイン6
stf007	社員7	シャイン7
stf008	社員8	シャイン8
stf009	社員9	シャイン9
stf010	社員10	シャイン10

<実行結果>

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

2.5 データの登録・更新・削除

データの参照ができるようになりましたので、データを追加登録するページを作成します。 ページの名称は staffins とし、前節で作成した staff.html から<IMART type="link">を使ってリンクを張ってください。



2.5.1 ベースとなるプレゼンテーション・ページ（.html）の作成

まず、社員データ登録用の一般的な HTML を記述します。データ送信を行うので、<FORM>タグを使います。次のリストを参考にして、ブラウザ上に正しく表示されるのを確認してください。

<データ更新フォーム表示用 HTML>

```
<HTML>
<BODY>
データを入力して登録ボタンを押してください。<BR>
  <FORM method="POST">
      社員コード
      <INPUT type="text" name="staff_code"><BR>
      社員名
      <INPUT type="text" name="staff_name"><BR>
      <BR>
      <INPUT type="submit" value="登録">
  </FORM>
  <BR>
  <IMART type="link" page="sample/user1/staff">戻る</IMART>
</BODY>
</HTML>
```

ブラウザ上に正しく表示されるのを確認後、intra-mart のファンクション・コンテナにデータを送るために、いくつかのタグを<IMART>タグに変更します。

<ファンクション・コンテナとの連携用 staffins.html>

```
<HTML>
<BODY>
  データを入力して登録ボタンを押してください。<BR>
  <IMART type="form" method="POST" action="insertStaffName">
      社員コード
      <IMART type="input" style="text" name="staff_code"></IMART><BR>
      社員名
      <IMART type="input" style="text" name="staff_name"></IMART><BR>
      <BR>
      <IMART type="submit" value="登録"></IMART>
  </IMART>
  <BR>
  <IMART type="link" page="sample/user1/staff">戻る</IMART>
</BODY>
</HTML>
```

2.5.2 ファンクション・コンテナ（.js）の作成

クライアント上のブラウザから送られてきたデータを、サーバ上のファンクション・コンテナ上でデータベースに反映するためのロジックを記述します。ファイル名は staffins.js です。

ここでは、次の2つのことを行います。

① クライアントからのリクエストデータの参照
② データベースへの追加

データベースへのレコード追加は、DatabaseManager オブジェクトに定義されている次の3つのメソッドを呼び出すことにより実現します。

beginTransaction メソッド	トランザクション開始の宣言
insert メソッド	データの挿入
commit/rollback メソッド	挿入したデータのコミット／ロールバック

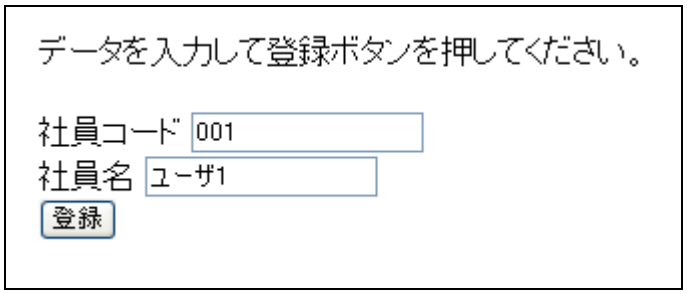
<データベースへのデータ登録用 Java スクリプト>

```
function insertStaffName(request)
{
    var objRecord = new Object();   // レコード登録用オブジェクトを生成
    var result;            // DBアクセスの結果

    // レコードの作成
    objRecord.staff_cd            = request.staff_code;
    objRecord.stf_name_kanji = request.staff_name;
    objRecord.stf_name_kana    = request.staff_name_kana;
    objRecord.stf_name_eng      = request.staff_name_eng;

    // DB処理の開始
    DatabaseManager.beginTransaction();

    // レコードの挿入
    result = DatabaseManager.insert("m_sample_stf", objRecord);

    // エラーチェック
    if (! result.error) {
        DatabaseManager.commit();                // 成功なのでコミットします
    }
    else{
        DatabaseManager.rollback();              // 失敗なのでロールバックします
    }
 }
```

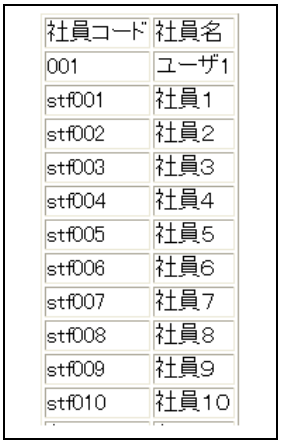

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

2.5.3 アプリケーションの実行

次に実行結果を示します。

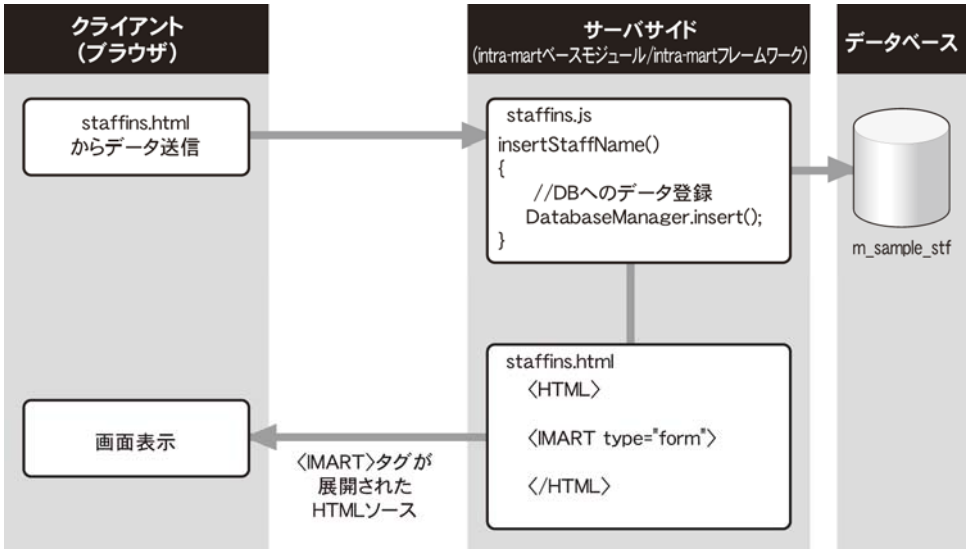


<staffins.html 実行画面>



社員コード	社員名
001	ユーザ1
stf001	社員1
stf002	社員2
stf003	社員3
stf004	社員4
stf005	社員5
stf006	社員6
stf007	社員7
stf008	社員8
stf009	社員9
stf010	社員10

<実行結果>



初期表示されている staffins.html は、intra-mart WebPlatform でタグ解釈されて送付された通常の HTML ファイルです。

■ 解説

◆ **DatabaseManager オブジェクト－2**

ここで利用されている、insert メソッドの他に、更新・削除用に次の2つのメソッドが用意されています。

メソッド名	機 能	パ ラ メ タ
updateメソッド	データの更新	update（表名，ＷＨＥＲＥ句の条件，更新データ）
removeメソッド	データの削除	remove（表名，ＷＨＥＲＥ句の条件）

パラメータに指定する「WHERE 句の条件」には、SQL 文で指定する WHERE 句に与える条件式を記述します。

例えば、「社員コードが stf0001 の社員を削除」したい場合は、

DatabaseManager.remove("m_sample_stf", "staff_cd = 'stf0001'");

のように指定します。実際には、プレゼンテーション・ページで指定した社員コードを参照する場合がほとんどですので、

```
var strWhere = "staff_cd = '" + request.staff_code + "'";
DatabaseManager.remove( "m_sample_stf", strWhere );
```

となります。

また、本オブジェクトでは、同時に複数のデータベースへのアクセスできるオプションも用意されています。

詳細は、「API リスト」の本オブジェクトの項を参照してください。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

3　Javaコンポーネント群（im-BizAPI）の利用

3.1　画面共通モジュール

Web ベースでの GUI 開発でよく利用される画面部品のモジュールです。それぞれのモジュールに適当なプロパティを設定して呼び出すだけで、データベースと連動したユーザインタフェースを簡単に作成できます。

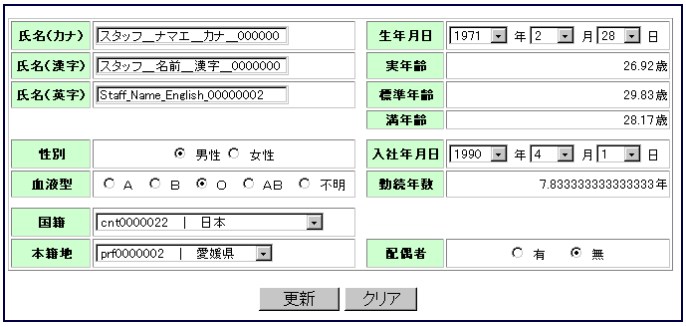
■　提供される画面共通モジュールの例

一般的な入力コントロール群	ユーザインタフェース構築に必要となる一般的な入力コントロール（テキストフィールド、パスワードボックス、ラジオボタン、チェックボックス、テキストエリア　など）を用意しています。これらのコントロール郡は、サーバサイドのスクリプトやデータと連動が可能なコントロールとなります。
レイアウト制御モジュール群	さまざまな条件により表示すべき値を変化させたり、表示する内容を選択したりするなど、HTMLでは表現できないプログラム的な要素をプレゼンテーション・ページ内に定義することができます。

■　構築された Web ユーザインタフェースの例

前述のオブジェクト／関数群を利用して HTML 上で編集していくことで、細かなレベルのユーザインタフェースの構築が可能になり、従来の VisualBasic などによるユーザインタフェースと遜色がないスタイルの Web システムの構築が可能です。

画面の作成例は以下のようになっています。



<画面の作成例>

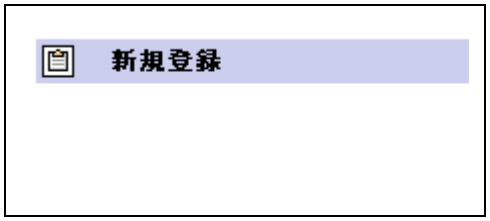
3.1.1　画面デザイン共通モジュール

<IMART>タグには、画面共通モジュールの他に、画面デザイン共通モジュールがあり、主に表示系に特化したタグ群が用意されています。

ここでは、例として、ImTitleBar タグを使ったサンプルを示します。ImTitleBar タグは、タイトルバーを自動生成します。

＜サンプル＞

```
<HTML>
    <HEAD>
        <IMART type="imDesignCss"></IMART>
    </HEAD>
    <BODY>
        <!-- タイトルバー表示 -->
        <IMART type="imTitleBar"
                title="新規登録"
                icon="images/standard/title.gif">
        </IMART>
    </BODY>
</HTML>
```



<実行画面>

この他にも、様々な画面デザイン共通モジュールが用意されています。詳細については API をご覧になってください。

Copyright 2000-2013 株式会社 NTT データイントラマート All rights Reserved.

3.1.2 標準画面の作り方（共通画面デザイン）

以下のドキュメントが用意されています。

- ◆ 画面デザインガイドライン
- ◆ スタイルシート仕様書

画面デザインガイドラインに準じた API も用意されています。API 仕様は、画面デザインガイドラインのドキュメントに掲載されています。

これを参考に、共通化 API を利用して画面を作成することにより、製品標準の画面と同様の画面デザインでアプリケーション開発ができます。デザインが共通化されると、他の画面と見た目や操作性が統一され、メニューから呼び出されたときに利用者が違和感無くアプリケーションを操作できるなどのメリットがありますので、アプリケーション開発の際には、デザイン統一の方法としてこのガイドラインの利用を検討してください。

※ intra-mart WebPlatform/AppFramework の持つ各画面ソースは、そのほとんどがプレーンな状態でインストールされています。画面デザインガイドラインの適用方法や、APIの使用例としてご活用ください。

3.2 Storage Serviceの利用方法

intra-mart ではStorage Service を利用することにより、分散システム構築時においても、容易にファイルの共有が可能となります。ここでは、ブラウザからアップロードされたファイルをStorage Service に保存したり、またStorage Service に保存されているファイルをダウンロードしたりする画面を作ってみます。

3.2.1 ファイル・アップロード

まずは、ファイルをアップロードしてStorage Service に保存するためのフォームを作成します。

<送信用フォーム「filer.html」>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">

 <HTML>
    <HEAD>
        <TITLE>File Center</TITLE>
    </HEAD>
    <BODY bgcolor="WhiteSmoke">
        <H2>Upload</H2>
        <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
            <INPUT type="file" name="local_file">
            <INPUT type="submit" value=" send ">
         </IMART>
     </BODY>
 </HTML>

 <!-- End of File -->
```

受信したファイルデータをStorage Service に保存するためのファンクション・コンテナを記述します。
ファンクション・コンテナでは、ファイルデータをバイナリ形式で受け取るので、Storage Service に対してもバイナリデータとしてファイル保存を行います。

<ファイルを受信するためのファンクション・コンテナ「filer.js」>

```
// 画面初期化
function init(request){

    var root = new VirtualFile("filebox");

    // 保存ディレクトリの有無のチェック
    if(! root.isDirectory()){
        root.makeDirectories(); // ディレクトリ作成
    }
}

// 指定ファイルの受信
function action_upload(request){

    // パラメータ情報(=RequestParameter オブジェクト)を取得
    var parameter = request.getParameter("local_file");

    // ファイルの中身を取得(バイナリ)
    var fileData = parameter.getValueAsStream();

    // ファイル名の取得
    var fileName = parameter.getFileName()

    // ファイルの書き出し
    var vf = new VirtualFile("filebox/" + fileName);
    var res = vf.save(fileData);
}
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

この画面では、以下のように動作します。

- ファイルコントロールにより選択されたファイルデータがサーバに送られます。
- サーバでは、ファイルの内容の有無を確認します。
- ファイルを受信していた場合、受信したファイルのファイル名を取得します。
- 元のファイル名のまま受信したファイルデータをStorage Service に出力します。

<実行画面（結果）>

■ 解説

- **<INPUT type="file">**

  HTML フォーム中で利用している<INPUT type="file">について、もう少し詳しく説明します。

  フォーム・コントロールである<INPUT type="file">を利用すると、ブラウザからサーバに対してファイルをアップロードすることができます。

  この時、フォームは以下のような記述が必要になります。

  <IMART type="form" method="POST" enctype="multipart/form-data">

  これは、ファイルの情報を MIME 形式にエンコードして POST モードでサーバにリクエストをするという指定になります。

- **RequestParameter オブジェクト**

  intra-mart では、アップロードされたファイルの操作を容易にする RequestParameter オブジェクトを用意しています。本オブジェクトを利用することで、サーバ側では受信したデータに対して特殊な処理を行うことなくファイルを取り出すことができます。RequestParameter オブジェクトには、アップロードされたファイルのファイル名を取得するメソッド等も用意されています。詳しくは API リストを参照してください。

3.2.2 ファイルリストの表示

ここでは、前節で作成した画面を改良して、アップロードしたファイルの一覧を表示するようにソースを修正します。

<一覧表を表示するためのプレゼンテーション・ページ（filer.html）>

```
<HTML>
    <HEAD>
        <TITLE>File Center</TITLE>
    </HEAD>
    <BODY bgcolor="WhiteSmoke">
        <H2>Upload</H2>
        <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
            <INPUT type="file" name="local_file">
            <INPUT type="submit" value=" send ">
        </IMART>
        <HR>
        <H2>File List</H2>
        <TABLE border="1">
            <TR><TH>File Name</TH></TR>
            <IMART type="repeat" list=fList item="rec">
                <TR>
                    <TD><IMART type="string" value=rec></IMART></TD>
                </TR>
            </IMART>
        </TABLE>
    </BODY>
</HTML>
```

アップロードされたファイルを一覧表として表示するためには、保存されているファイルをリストとして取得する必要があります。

Storage Service に問い合わせてファイルリストを取得するためのコードを追加します。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

<ファイル一覧を取得するように修正したファンクション・コンテナ>

```
var fList, rec;

// 画面初期化
function init(request){

	var root = new VirtualFile("filebox");

	// 保存ディレクトリの有無のチェック
	if(! root.isDirectory()){
		root.makeDirectories();
	}

	fList = root.files(); // ファイルリストの取得
}

// 指定ファイルの受信
function action_upload(request){

	// パラメータ情報(=RequestParameter オブジェクト)を取得
	var parameter = request.getParameter("local_file");

	// ファイルの中身を取得（バイナリ）
	var fileData = parameter.getValueAsStream();

	// ファイル名の取得
	var fileName = parameter.getFileName()

	// ファイルの書き出し
	var vf = new VirtualFile("filebox/" + fileName);
	var res = vf.save(fileData);

}
```

■ 解説

◆ **Storage Service 上のファイルを操作する API VirtualFile**

Storage Service(標準では%Storage Service%/storage/)にあるファイルやディレクトリの作成、削除、および各種情報の取得を行う API です。

この API を利用することで様々なファイルの作成や取得、またディレクトリの作成やリストの取得などができ、分散システムにおいても、すべての Application Runtime 間で、環境に依存することなくファイルを共有することができます。

<実行画面（結果）>

この実行画面では、すでに 1 つのファイルがアップロードされていることが確認できます。

3.2.3 ファイル・ダウンロード

ファイルをアップロードできて、アップロードされたファイルがStorage Service に保存されていることが確認できたので、次は、保存されているファイルをブラウザにダウンロードする機能を追加します。前節で作成した画面に対してファイルをダウンロードできるようにソースを修正します。

ファイルをダウンロードするには、プレゼンテーション・ページにダウンロードをするためのリンクを追加します。また、そのリンクがクリックされた時にファイルを送信するためのロジックをファンクション・コンテナに追加していきます。

<ダウンロード用リンクを追加したプレゼンテーション・ページ(filer.html)>

```
<HTML>
    <HEAD>
        <TITLE>File Center</TITLE>
    </HEAD>
    <BODY bgcolor="WhiteSmoke">
        <H2>Upload</H2>
        <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
            <INPUT type="file" name="local_file">
            <INPUT type="submit" value=" send ">
        </IMART>
        <HR>
        <H2>File List</H2>
        <TABLE>
            <TR><TH>File Name</TH><TH></TH></TR>
            <IMART type="repeat" list=fList item="rec">
                <TR>
                    <TD><IMART type="string" value=rec></IMART></TD>
                    <TD>
                      <IMART type="link" action="action_download" server_file=rec>download</IMART>
                    </TD>
                </TR>
            </IMART>
        </TABLE>
    </BODY>
</HTML>
```

download と表示するためのリンクを作成します。
このリンクには、クリックされた時にダウンロード処理をするための関数 action_download が指定されていますので、ファンクション・コンテナ内にダウンロード関数 action_download を追加します。

<ダウンロード用関数を追加したファンクション・コンテナ>

```
var fList, rec;

// 画面初期化
function init(request){
    var root = new VirtualFile("filebox");

    // 保存ディレクトリの有無のチェック
    if(! root.isDirectory()){
        root.makeDirectories();
    }

    fList = root.files(); // ファイルリストの取得
}

// 指定ファイルの受信
function action_upload(request){

    // パラメータ情報(=RequestParameter オブジェクト)を取得
    var parameter = request.getParameter("local_file");

    // ファイルの中身を取得(バイナリ)
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

```
        var fileData = parameter.getValueAsStream();

        // ファイル名の取得
        var fileName = parameter.getFileName()

        // ファイルの書き出し
        var vf = new VirtualFile("filebox/" + fileName);
        var res = vf.save(fileData);

}

 // 指定ファイルの送信
 function action_download(request){
        var fpath = new VirtualFile("filebox/" + request.server_file); // 取得
        Module.download.send(fpath.load(), request.server_file);          // 送信
 }
```

<実行画面（結果）>

■ 解説

◆ ダウンロード API Module.download.send()

サーバから HTML 形式以外での情報の送信を行うための API です。主に、ファイルをダウンロードする場合に利用します。情報をダウンロードする時には、ブラウザが、その情報はどのような形式なのかを判別するための情報を付加する必要がありますが、この API では、自動的に情報の形式を判断して適切な形でダウンロードを行える機能を提供しています。

例えば、Word で保存されたファイルデータをダウンロードする場合、ファイル名として『 ***.doc 』という拡張子を持つ名称を与えます。ダウンロード API では、ファイルの拡張子を判断して MIME コードを決定しますので、ダウンロードしたコンピュータに Word がインストールされている場合には、ブラウザ内にドキュメントが表示されますし、 Word がインストールされていない場合には、ファイルを保存するためのダイアログボックスが表示されます。

3.2.4 ファイルの削除

前節までで、ファイルをアップロードすることと、アップロードしたファイルをダウンロードすることができました。しかし、このままではアップロードされたファイルがStorage Service に溜まっていってしまうので、ここでは、アップロードされてStorage Service 上に保存されているファイルを削除する機能を追加します。

<削除用リンクを追加したプレゼンテーション・ページ（filer.html）>

```
<HTML>
    <HEAD>
        <TITLE>File Center</TITLE>
    </HEAD>
    <BODY bgcolor="WhiteSmoke">
        <H2>Upload</H2>
        <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
            <INPUT type="file" name="local_file">
            <INPUT type="submit" value=" send ">
        </IMART>
        <HR>
        <H2>File List</H2>
        <TABLE>
            <TR><TH>File Name</TH><TH></TH></TR>
            <IMART type="repeat" list=fList item="rec">
                <TR>
                    <TD><IMART type="string" value=rec></IMART></TD>
                    <TD>
                     <IMART  type="link"    action="action_download"
                         server_file=rec>download</IMART>
                         <IMART type="link" action="action_remove"
                         server_file=rec>remove</IMART>
                    </TD>
                </TR>
            </IMART>
        </TABLE>
    </BODY>
</HTML>
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

<削除用関数を追加したファンクション・コンテナ（filer.js）>

```
var fList, rec;

// 画面初期化
function init(request){
	var root = new VirtualFile("filebox");

	// 保存ディレクトリの有無のチェック
	if(! root.isDirectory()){
		root.makeDirectories();
	}

	fList = root.files(); // ファイルリストの取得
}

// 指定ファイルの受信
function action_upload(request){

	// パラメータ情報(=RequestParameter オブジェクト)を取得
	var parameter = request.getParameter("local_file");

	// ファイルの中身を取得（バイナリ）
	var fileData = parameter.getValueAsStream();

	// ファイル名の取得
	var fileName = parameter.getFileName()

	// ファイルの書き出し
	var vf = new VirtualFile("filebox/" + fileName);
	var res = vf.save(fileData);

}

// 指定ファイルの送信
function action_download(request){
	var fpath = new VirtualFile("filebox/" + request.server_file); // 取得
	Module.download.send(fpath.load(), request.server_file);		// 送信
}

// 指定ファイルの削除
function action_remove(request){
	var fpath = new VirtualFile("filebox/" + request.server_file); // 取得
	fpath.remove();  // 削除
}
```

<実行画面（結果）>

削除リンクをクリックすると、該当するファイルを削除することができます。

■ 解説

◆ **VirtualFile の remove()メソッド**

ファイルを削除するための API です。

この API では、ファイルの他にディレクトリも削除することができます。

ただし、ディレクトリを削除する場合には、削除対象としているディレクトリ内にファイルやディレクトリが存在せずに空である必要がありますので注意してください。

画面遷移について

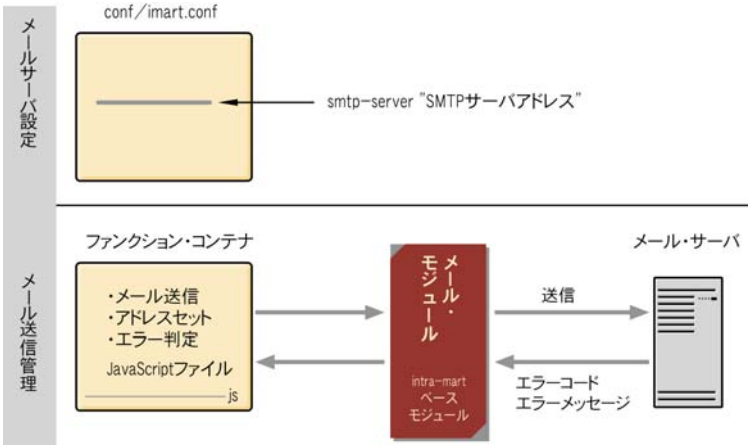
このサンプルでは、画面遷移を一切行わずに 1 画面ですべての機能を持っています。

この場合、各機能に対するリクエストや、リクエスト後の画面の表示の際に、リンクやフォームに対する page 指定を行わなくても、自分自身を再度表示することができます

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.3 メール連携モジュール（ファンクション・コンテナ）

本モジュールを利用することで、SMTP/POP3 互換のメールサーバに対するメールの送信処理を行うことができます。



3.3.1 メール送信

ここでは、メールを送信するための画面を作成します。

メールを送信するためには、conf/imart.xml 内の SMTP サーバの設定を正しく行ってください。

■ メールサーバの設定

メールサーバの設定は、conf/imart.xml ファイルで行います。記述例は以下の通りです。

<メールサーバの設定>

```
<smtp-server host="localhost" port="25" mailbox-check="false" />
```

■ メール送信管理の設定

メール送信の設定には、MailSender オブジェクトを用います。MailSender オブジェクトのメソッドは以下の 11 より構成されており、これらを用いてファンクション・コンテナでメール送信の設定を行います。

```
(1)  setFrom(String address ,String personal)     ：メール送信元(From)を設定するメソッド
(2)  addTo(String address ,String personal)  ：メール送信先(To)を追加するメソッド
(3)  addCc(String address ,String personal)  ：メール送信先(Cc)を追加するメソッド
(4)  addBcc(String address ,String personal)：メール送信先(Bcc)を追加するメソッド
(5)  addReplyTo(String replyto)     ：メール返信先を追加するメソッド
(6)  addHeader(String name ,String value)   ：メールヘッダーを追加するメソッド
(7)  setSubject(String subject)     ：メール題名(Subject)を設定するメソッド
(8)  setText(String text)：本文を設定するメソッド
(9)  addAttachment(String filename ,String file)  ：メールへの添付ファイルを追加するメソッド
(10) send()：メールを送信するメソッド
(11) getErrorMessage()    ：メール送信エラー時のメッセージを取得するメソッド
```

前記メソッドの記述例は以下のようになります。

<メールの送信処理（ファンクション・コンテナ）>

```
var ret;
var errorMessage;

var locale = AccessSecurityManager.getSessionInfo().locale;   // ロケールの取得
var mailSender = new MailSender(locale);     // MailSender オブジェクトを生成

      //-----------------------------
// 送信情報の設定
//-----------------------------

// 送信先メールアドレス
mailSender.addTo("mail000@nttdata.co.jp");
mailSender.addTo("mail001@nttdata.co.jp");
mailSender.addTo("mail002@nttdata.co.jp");
mailSender.addTo("mail003@nttdata.co.jp");
mailSender.addTo("mail004@nttdata.co.jp");

// CC メールアドレスをセット
mailSender.addCc("mail005@nttdata.co.jp");

// 送信元メールアドレス
mailSender.setFrom(request.mail_from);

//-----------------------------
//メールタイトルと内容をセット
//-----------------------------

// 題名の設定
mailSender.setSubject("メール送信サンプル");

// 本文の設定
mailSender.setText("メール送信のテストです。" + "¥n" + "うまく送れましたか？");

//メール送信
ret = mailSender.send();

      //エラー判定
if( ret ) {
      errorMessage = "エラーメッセージ：" + mailSender.getErrorMessage();
            //メール送信エラー
            Module.alert.back( "SYSTEM.ERR", errorMessage);
}
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.3.1.1 メール送信フォームの作成

メールを送信するためのフォームを作成します。

プレゼンテーション・ページのフォーム内には、メール送信に必要な送信先アドレス、送信者アドレス、件名、本文を登録するコントロールを用意します。また、ファンクション・コンテナでは、受け取った情報を元にして MailSender API を利用してメール送信を行うための関数を定義します。

<メール送信用フォームを記述したプレゼンテーション・ページ（sender.html）>

```
<HTML>
    <HEAD>
        <TITLE>Mail Sender</TITLE>
    </HEAD>
    <BODY bgcolor="WhiteSmoke">
        <IMART type="form" action="action_send" method="POST">
            To: <INPUT type="text" name="mail_to"><BR>
            From: <INPUT type="text" name="mail_from"><BR>
            Subject: <INPUT type="text" name="mail_subject"><BR>
            Message: <TEXTAREA name="mail_body" cols="40" rows="8"></TEXTAREA><BR>
            <INPUT type="submit" value=" send ">
        </IMART>
    </BODY>
</HTML>
```

<メール送信ロジックを記述したファンクション・コンテナ（sender.js）>

```
// メールの送信
function action_send(request){

    // ロケールの取得
    var locale = AccessSecurityManager.getSessionInfo().locale;

    // MailSender オブジェクトを生成
    var mailSender = new MailSender(locale);

    // 送信情報の設定
    mailSender.addTo(request.mail_to);
    mailSender.setFrom(request.mail_from);
    mailSender.setSubject(request.mail_subject);

    // 本文の設定
    mailSender.setText(request.mail_body);

    // メールの送信
    if( mailSender.send() ){
        // 送信成功
        Module.alert.link("SYSTEM.SUCCESS",
                          "メールを送信しました。", "sender");
    }
    else{
        // 送信失敗
        Module.alert.link("SYSTEM.ERR",
                          mailSender.getErrorMessage(),"sender");
    }

}
```

<実行画面(結果)>

フォーム中の必要事項をすべて入力した後に[send]ボタンをクリックするとメールを送信することができます。

■ 解説

◆ **MailSender API**

メールを送信するための API です。

サンプルでは、送信先、送信者、件名、本文の設定しか行っていませんが、CC や BCC の設定をすることもできます。

◆ **TO および From 設定方法**

MailSender オブジェクトを利用することで、送信先や送信者、また CC や BCC などの設定には、メールアドレスだけではなく名前も設定することができます。詳しくは API リストを参照してください。

◆ **メール送信とサーバ処理速度**

メールを送信する場合、Application Runtime と SMTP サーバが連携する必要があります。

送信するメールの情報量はもちろんのこと、ネットワーク環境やネットワークトラフィックなどによりメール送信処理時間がかかる場合があります。

3.3.2 添付ファイル付きメールの送信

前節で作成したメール送信フォームを改良して添付ファイルをメール本文と共に送信できるようにします。プレゼンテーション・ページでは添付ファイルをアップロードするためのフォーム・コントロールを追加し、フォームの属性を変更します。

ファンクション・コンテナは、フォームの修正に合わせて、メール送信関数を添付ファイルに対応できるように修正します。

<ファイルアップロード用コントロールを追加したプレゼンテーション・ページ(sender.html)>

```
<HTML>
  <HEAD>
      <TITLE>Mail Sender</TITLE>
  </HEAD>
  <BODY bgcolor="WhiteSmoke">
     <IMART type="form" action="action_send" method="POST" enctype="multipart/form-data">
          To: <INPUT type="text" name="mail_to"><BR>
          From: <INPUT type="text" name="mail_from"><BR>
          Subject: <INPUT type="text" name="mail_subject"><BR>
          Attachment: <INPUT type="file" name="mail_file"><BR>
          Message: <TEXTAREA name="mail_body" cols="40" rows="8"></TEXTAREA><BR>
          <INPUT type="submit" value=" send ">
   </IMART>
  </BODY>
</HTML>
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

■ 解説

◆ ファイルをアップロードするためのフォーム

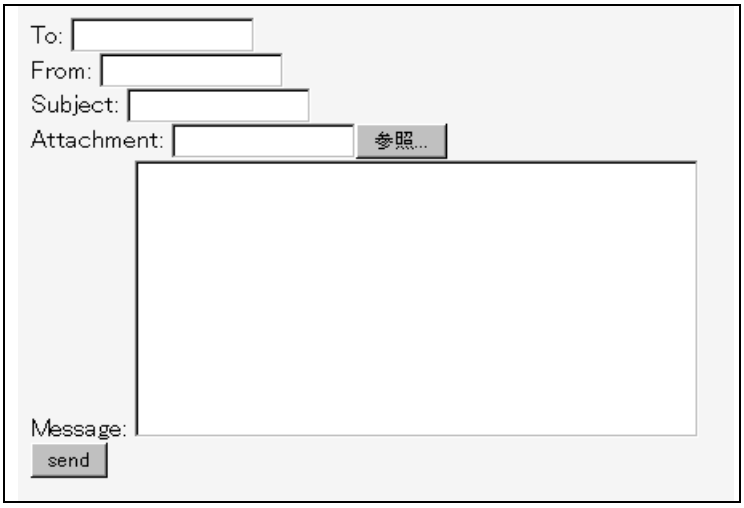
ファイルをアップロードするためには以下のようなフォームの記述が必要になります。

<IMART type="form" method="POST" enctype="multipart/form-data">

こうすることで、サーバ上では、フォーム・コントロール <INPUT type="file"> によりローカルのファイルを受け取ることができます。

<添付ファイル送信に対応したファンクション・コンテナ（sender.js）>

```
// メールの送信
function action_send(request){

        // ロケールの取得
        var locale = AccessSecurityManager.getSessionInfo().locale;

        // MailSender オブジェクトを生成
        var mailSender = new MailSender(locale);

        // 送信情報の設定
        mailSender.addTo(request.mail_to);
        mailSender.setFrom(request.mail_from);
        mailSender.setSubject(request.mail_subject);

        // 添付ファイルの設定(RequestParameter オブジェクトとして取得)
        var parameter = request.getParameter("mail_file");

        if( parameter != null && parameter.getLength() > 0 ) {

         // ファイル名の取得
        fileName = parameter.getFileName()

        // ファイルの中身を取得（バイナリ）
        fileData = parameter.getValueAsStream();

        添付ファイルの設定
        mailSender.addAttachment(fileName, fileData);

   }

        // 本文の設定
         mailSender.setText(request.mail_body);

        // メールの送信
        if( mailSender.send() ){
              // 送信成功
              Module.alert.link("SYSTEM.SUCCESS",
                                        "メールを送信しました。","sender");
        }
        else{
              // 送信失敗
              Module.alert.link("SYSTEM.ERR",
                                        mailSender.getErrorMessage(),"sender");
        }
}
```



<実行画面(結果)>

■ 解説

◆ 添付ファイルとメール送信速度

ファイルを添付してメール送信する場合、RFC の規約によりファイルデータそのものをエンコードしたのち、メール本文を含めたメール情報全体をエンコードしてから SMTP サーバに対して送信する必要があります。このデータのエンコード処理は MailSender API が自動的に行いますが、エンコード処理時には、アプリケーションサーバに負荷が掛かります。

◆ 添付ファイルと処理速度

添付ファイルを送信する場合、ブラウザが Web サーバに対してファイルデータを送信し、その情報を受信したアプリケーションサーバがメール送信処理を行います。

1 つのメール送信に対して複数のネットワークを介しますので、サイズの大きなファイルを添付してメール送信する場合には、メール送信処理に時間がかかってしまう場合があります

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.4 共通ライブラリの作成

intra-mart WebPlatform では、ユーザが<IMART>タグや、グローバル関数、API などを自由に定義し、利用することが可能です。この章では、これらの設定方法や利用方法などを紹介します。

3.4.1 拡張APIの作成

intra-mart WebPlatform では、API に関してもユーザが定義することが可能です。ユーザ定義の拡張 API はユーザ独自のオブジェクトを実装した js ファイルと system_install.xml によって登録することができます。
オブジェクトの実装の仕方によって、静的な関数を定義した API、および new 演算子を用いてインスタンスを生成する形式の API を実装することができます。

3.4.1.1 API の登録

■ 設定方法(静的なメソッド)

1. この例では、静的な関数を持つオブジェクトを登録します。作成した js ファイルは任意の場所に保存してください。

＜ソース例＞

```
/**
 * static オブジェクトの作成
 */
function ImSampleStaff(){}

/**
 * すべてのスタッフコードを取得する
 */
ImSampleStaff.getStaffCds = function (){
   var sql =  "SELECT staff_cd FROM m_sample_stf "
   var result = DatabaseManager.execute(sql);
   return result;
}

/**
 * すべてのスタッフ名を取得する
 */
ImSampleStaff.getStaffNames = function (){
   var sql =  "SELECT stf_name_kanji FROM m_sample_stf "
   var result = DatabaseManager.execute(sql);
   return result;
}
```

2. conf/system_install.xml ファイルに以下の内容を加えます。記述方法は、<api-script>オブジェクトを定義した js ファイルパス#オブジェクト</api-script>です。

```
<system-install>
  …
  <java-script-api>
  …
        <api-script>sample/prog_guide/common_libs/api/im_sample_staff#ImSampleStaff</api-script>
```

3. サーバを再起動してください。

4. API を利用するには、ファンクション・コンテナからの呼び出しを行います。

＜ソース例＞

```
var aCdList = new Array();
var aNameList = new Array();

function init(){
   aCdList = getCdList().data;
   aNameList = getNameList().data;
}

function getCdList(){
   var oData =   ImSampleStaff.getStaffCds();
   return oData;
}

function getNameList(){
   var oData = ImSampleStaff.getStaffNames();
   return oData;
}
```

＜ソース例＞

```
<HTML>
<HEAD><TITLE>test</TITLE>
</HEAD>
<BODY>
<TABLE>
   <TD>
   [staff cd list]<BR><BR>
   <TABLE border="1">
         <IMART type="repeat" list=aCdList item="record" index="idx">
         <TR>
               <TD><IMART type="string" value=record.staff_cd></IMART></TD>
         </TR>
         </IMART>
   </TABLE>
   </TD>
   <TD>
   [staff name list]<BR><BR>
   <TABLE border="1">
         <IMART type="repeat" list=aNameList item="record" index="idx">
         <TR>
               <TD><IMART type="string" value=record.stf_name_kanji></IMART></TD>
         </TR>
         </IMART>
   </TABLE>
   </TD>
</TABLE>
</BODY>
</HTML>
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

<実行結果>

■ 設定方法(インスタンス オブジェクト)

1. この例では、インスタンスな関数を持つオブジェクトを登録します。作成した js ファイルは任意の場所に保存してください。

<ソース例(im_area_calc.js)>

```
/**
 * コンストラクタ
 */
function ImAreaCalc(valueA, valueB, areaType) {
	this.valueA     = valueA;
	this.valueB     = valueB;
	this.areaType = areaType;

	this.getArea    = _getAreaFunction;
}

/**
 * 面積を求める関数
 */
function _getAreaFunction() {

	switch(this.areaType) {
		case "square" :
			return    this.valueA * this.valueB;

		case "triangle" :
			return (this.valueA * this.valueB) / 2;

		default :
			return;
	}

}
```

2. system_install.xml ファイルに以下の内容を加えます。記述方法は、<api-script>オブジェクトを定義した js ファイルパス#オブジェクト</api-script>です。

```
<system-install>
  …
</java-script-api>
…
  <api-script>sample/prog_guide/common_libs/api/im_area_Calc#ImAreaCalc</api-script>
```

3. サーバを再起動してください。

4. API を利用するには、ファンクション・コンテナからの呼び出しを行います。

＜ソース例＞

```
var square;
var triangle;

function init(){
 var area = new ImAreaCalc(8,3,"square");
 square = area.getArea();
 area = new ImAreaCalc(8,3,"triangle");
 triangle = area.getArea();
}
```

＜ソース例＞

```
<HTML>
<HEAD><TITLE></TITLE>
</HEAD>
<BODY>
  縦 8cm 横 3cm の四角形の面積 :
  <IMART type="string" value=square></IMART><br>
  底辺 8cm 高さ 3cm の三角形の面積:
  <IMART type="string" value=triangle></IMART><br>
</BODY>
</HTML>
```

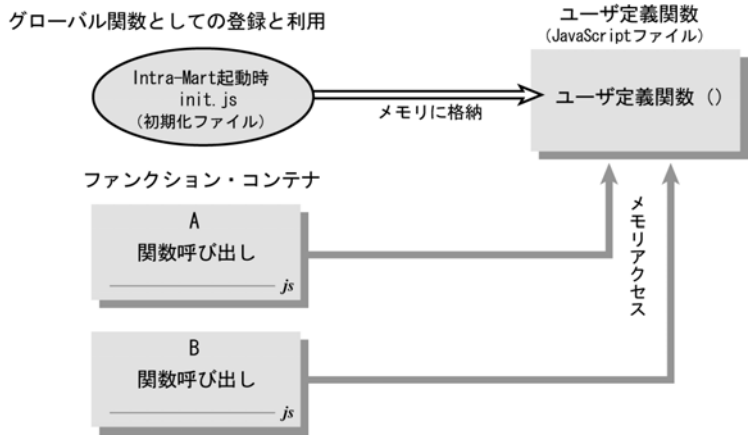
縦 8cm 横 3cm の四角形の面積：24
底辺 8cm 高さ 3cm の三角形の面積: 12

<実行結果＞

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.4.2 グローバル関数の作成

intra-mart WebPlatform には JavaScript で記述したユーザ定義関数を「グローバル関数」として登録する方法があります。グローバル関数と、ファンクション・コンテナからの呼び出しの構造は以下のようになっています。



<ユーザ定義関数の利用形態>

グローバル関数には、Procedure.define 関数と system_install.xml 設定ファイルの 2 通りの定義方法があります。

3.4.2.1 sysem_install.xml 設定ファイル

様々な共通ライブラリ情報などを設定するファイルが system_install.xml です。グローバル関数が定義されているファイルの指定をすることで、Procedure.define()関数や include()関数を用いずとも、グローバル関数を使用することができます。

■ 設定方法

1. グローバル関数として登録する js ファイルを作成します。作成した js ファイルは任意の場所に保存してください。

＜ソース例＞

```
//共通関数作成
function imSubVariables( valueA, valueB )
{
        return valueA - valueB;
}
```

2. conf/system_install.xml ファイルに以下の内容を加えます。記述方法は<global-function-script>グローバル関数を定義した js ファイルのパス#実行関数</tag-script>となっています。

＜ソース例＞

```
<system-install>
  …
  <java-script-api>
  …
        <global-function-script>sample/prog_guide/common_libs/global/sub_variables#imSubVariables
        </global-function-script>
```

3. グローバル関数の利用。
ファンクション・コンテナ内からグローバル関数の呼び出しを行います。

＜ソース例＞

```
function sub(){
   res = imSubVariables(10,3);
  return res;
}
```

3.4.2.2 Procedure.define 関数

■ Procedure define

◆ メソッド

```
void define (String name ,Object func)
```

JS関数（オブジェクト等）を設定します。

◆ パラメーター

String name	設定プロパティ名称（文字列）
Object func	JS 関数

■

※ Procedure.define() メソッドを使い、「グローバル関数」として登録すると、多少メモリを消費しますが、関数をメモリに保持するため実行速度は速くなります。

■ ※ Procedure.define() メソッドの詳細については、API リスト「アプリケーション共通モジュール」の「Procedure」を参照してください。

■ 設定方法

ユーザ定義関数を任意の js ファイルに作成し、メモリ上に格納されるよう初期化ファイル（pages/src/init.js）に記述をします。これによって、intra-mart 起動時にユーザ定義関数がメモリ上に格納され、ファンクション・コンテナからダイレクトにメモリに呼び出しがかかるようになります。ユーザ定義関数の呼び出しは、ファンクション・コンテナに記述を行います。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

1. 任意の js ファイルにユーザ定義関数を格納する

ここでは例として、「sample/prog_guide/common_libs/global/add_variables.js ファイル」に共通関数「addVariables()」を作るものとします。グローバルユーザ定義関数を格納する js ファイルに以下のように、Procedure.define()メソッドを使用した記述をします。

＜ソース例(add_variables.js)＞

```
//ユーザ定義関数を js ファイルへ格納する記述
//共通関数として登録する
Procedure.define("addVariables", addVariables);

//共通関数作成
function addVariables( valueA, valueB ) {
   return valueA + valueB;
}
```

2. 起動時にメモリ上に格納される設定を行う

ユーザ定義関数を格納した js ファイルについて、intra-mart 起動後にメモリ上に格納されるよう pages/src/init.js ファイル（初期化ファイル）にその js ファイルを取り込む記述をします。

＜ソース例＞

```
//共通関数格納ファイルを取り込む記述
/* init.js */

//共通関数格納ファイルの取り込み
include("sample/prog_guide/common_libs/global/add_variables");
```

- ※ 必ずしも pages/src/init.js 内に記述しなくても、他の JavaScript ファイルから一度 include()で取り込まれた関数は、以後グローバル関数として使用することができます。

3. ファンクション・コンテナにユーザ定義関数を呼び出す記述をする

ユーザ定義関数を必要とするファンクション・コンテナに、intra-mart 起動後メモリ上に格納されたその関数を呼び出す処理を記述をします。

ここでは例として、ファンクション・コンテナの「sample/prog_guide/common_libs/global/add_val_exe.js ファイル」で共通関数「addVariables()」を呼び出すものとします。
ユーザ定義関数を呼び出すには、アプリケーション js ファイルに、以下のように記述します。

＜ソース例＞

```
//ファンクション・コンテナにてグローバルユーザ定義関数を呼び出す記述

/* sample/prog_guide/common_libs/global/add_val_exe.js */

function add( ) {
   //共通関数の呼び出し
   return Procedure.addVariables( 1, 2 );
}
```

3.4.3 拡張<IMART>タグの作成

intra-mart 側で既に用意されている<IMART>タグ以外に、ユーザが定義し、独自の機能を持たせた任意の<IMART>タグをを拡張<IMART>タグと言います。
具体的には、<IMART>タグに対して、新しい type 属性を定義し、その type 属性が要求された時に、実際に処理を実行するタグ関数を登録することで、プレゼンテーション・ページ内で利用できる API を拡張できる機能です。
（<IMART type="xxx">の xxx の部分をユーザ独自に定義することができます。）

3.4.3.1 Imart.defineType 関数

拡張<IMART>タグは Imart.defineType 関数によって、定義することが出来ます。Imart.defineType 関数の書き方には 2 通りあり、初期起動時に実行できる場所(拡張<IMART>タグが呼び出される前に実行されるファイル)に記述する方法と、system_install.xml 設定ファイルに記述する方法があります。

■ Imart defineType

◆ メソッド

```
static void defineType (String name ,Function func)
```

◆ パラメータ

String name	定義名称(<IMART>タグの type=""に該当する値)
Function func	実行関数

■ 設定方法（初期起動時実行）

Imart.defineType()関数を記述する場所の一つに、初期起動時の実行ファイルがあげられます。これは pages/src/init.js などが挙げられ、サーバ起動時に一回だけ実行されるファイルを指します。そこに Imart.defineType()関数を記述し、実行されると、Imart オブジェクトに対して情報が登録され、以降、プレゼンテーション・ページ内では、いつでも利用可能になります。

1. 拡張<IMART>タグの実装内容と、Imart.defineType()関数を pages/src/init.js に書き加えます。

＜init.js＞

```
Imart.defineType("imEchoLoop",im_echo_loop);

function init(request) {
}

function im_echo_loop(oAttr,oInner){
   var time = new Number(oAttr.time);
   var str = oAttr.str;
   var index = new Array();
   var res = "";

   for(var i=0;i<time;i++){
         res += str;
         res += "<br>";
   }

   return res;
}
```

サーバ起動時に、この Imart.defineType()関数が呼ばれ、拡張<IMART>タグが登録されます。
この拡張<IMART>タグ実行関数が返却した文字列が、HTML ソースとしてブラウザに送信されることになります。
また、必ず文字列を返却しなければなりません。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

2. 拡張<IMART>タグの利用
登録した拡張<IMART>タグを使用するには、プレゼンテーション・ページ内でタグの呼び出しを記述します。

＜ソース例＞

```
<!-- echo_loop_sample.html -->
<html>
<head><title>echo test</title>
</head>
<body>
  拡張 タグ imEchoLoop の実行<BR>
  <BR>
      <IMART type="imEchoLoop" str="hello" time="8"></IMART>
</body>
</html>
```

拡張 タグ imEchoLoopの実行

hello
hello
hello
hello
hello
hello
hello
hello

＜実行結果＞

※ 詳細については、API リスト「アプリケーション共通モジュール」の「Imart.defineType()」を参照してください。

■ 設定方法(system_install.xml 設定ファイル)
様々な共通ライブラリ情報などを設定するファイルが system_install.xml です。ここに<IMART>タグの実装をしているファイルの指定をすることで、初期起動時に実行する場所に限らず、任意の場所に拡張<IMART>タグの実装ファイルを置くことができます。

1. 拡張<IMART>タグを実装する js ファイルを作成します。

＜echo_loop.js＞

```
function im_echo_loop(oAttr,oInner){
  var time = new Number(oAttr.time);
  var str = oAttr.str;
  var index = new Array();
  var res = "";

  for(var i=0;i<time;i++){
      res += str;
      res += "<br>";
  }

  return res;
}
```

2. 作成した拡張<IMART>タグをサーバに登録するために、conf/system_install.xml ファイルに以下の内容を加えます。記述方法は<tag-script>タグ定義 js ファイルのパス#実行関数</tag-script>となっています。

＜system_install.xml＞

```
<system-install>
        …
        <jssp-tag>
            …
        <tag-script>sample/prog_guide/common_libs/tag/echo_loop#im_echo_loop</tag-script>
```

3. サーバを再起動してください。

4. 登録した拡張<IMART>タグを使用するには、プレゼンテーション・ページ内でタグの呼び出しを記述します。

＜ソース例＞

```
<!-- echo_loop_sample.html -->
<html>
<head><title>echo test</title>
</head>
<body>
  拡張 タグ imEchoLoop の実行<BR>
  <BR>
        <IMART type="imEchoLoop" str="hello" time="8"></IMART>
</body>
</html>
```

拡張 タグ imEchoLoopの実行

hello
hello
hello
hello
hello
hello
hello
hello

<実行結果>

■ 解説

◆ **<IMART>タグへの定数値の設定**

あらかじめユーザが設定した定数または関数を<IMART>タグの属性で指定したキーワードで呼び出すことができます。詳細は、「API リスト」の Imart オブジェクトの次のメソッドを参照してください。

Imart.defineAttribute(sKeyWord, value)

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.5 外部プロセスの呼び出し

ユーザが作成したプログラムを intra-mart アプリケーションから実行するには、アプリケーション共通モジュールのグローバル関数「execute()」を利用します。この関数は、指定された文字列コマンドを新しいプロセスとして実行し、実行したプロセスが終了するまでこの関数は待機状態になります。

オブジェクト型関数	（Object） execute（（String） command）
入力値	（String） command：実行するコマンド
返却値	返却値はオブジェクト型で以下の形式になります。

＜ソース例＞

```
var message = "";

function init(request) {
   var oReturn = execute("ipconfig");
   message = oReturn.output;
}
```

```
Windows IP Configuration Ethernet adapter ローカル エリア接続:
Connection-specific DNS Suffix . : IP Address. . . . . . . . . . . :
192.168.108.73 Subnet Mask . . . . . . . . . . . : 255.255.254.0 Default
Gateway . . . . . . . . . : 192.168.108.1
```

<実行画面>

＜実行したプロセスが正常終了した場合＞

```
return_object
      ├ output // プロセスからの標準出力ストリーム
          （String）
      ├ error // プロセスからのエラー出力ストリーム
          （String）
      └ exit // プロセスの終了コード
```

＜実行したプロセスが正常終了しなかった場合＞

```
return_object
      ├ error //エラー内容（String）
      └ exit //プロセスの終了コード
```

※ プロセスの終了コードは、0 の場合正常終了となります。

※ 詳細は、API リスト「アプリケーション共通モジュール」のグローバル関数「execute()」を参照してください。

3.6 JavaClassとの連携

intra-mart で使用しているサーバサイド JavaScript には、さまざまな優れた機能が実装されています。しかし、スクリプト言語としての制限から通信機能の実現や、特殊なファイルアクセス等、システム構築上問題となる場合があります。

このようなシステム構築において問題となる部分を拡張する機能として、JavaScript と JavaClass の連携機能を説明します。

3.6.1 標準JavaClassとの連携方法

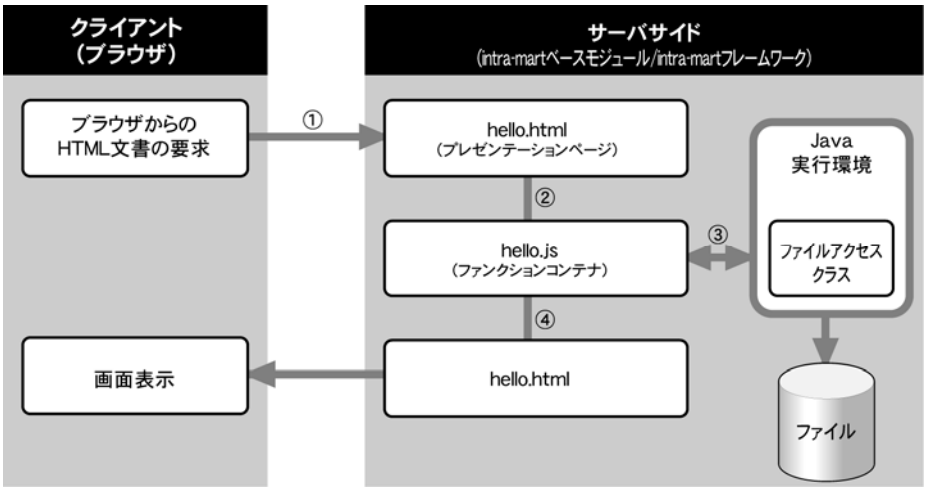
intra-mart は JavaVM 上で動作しており、intra-mart から簡単に Java の標準 Class と連携を行うことができます。

intra-mart からは、決められた宣言方法を用いてクラスを定義することにより、intra-mart のオブジェクトと同等に JavaClass メソッドにアクセスすることができます。

前項で説明した「HelloWorld」アプリケーションを例に JavaClass を用いた方法について説明します。

この Java 版の「HelloWorld」アプリケーションは、サーバ上にあるファイルの中身を表示するアプリケーションです。

本アプリケーションの処理の流れについて説明します。



<「HelloWorld」アプリケーション Java 版の処理イメージ>

1. Web ブラウザからサーバ上のプレゼンテーション・ページ（HTML）ファイル（hello.html）を起動する。

<hello.html（プレゼンテーション・ページ）>

```
<HTML>
<BODY>
こんにちは、<IMART type="string" value=nameValue></IMART>です。</H1>
</BODY>
</HTML>
```

2. プレゼンテーション・ページと連動したファンクション・コンテナ（hello.js）であるサーバサイド JavaScript の処理が開始される。

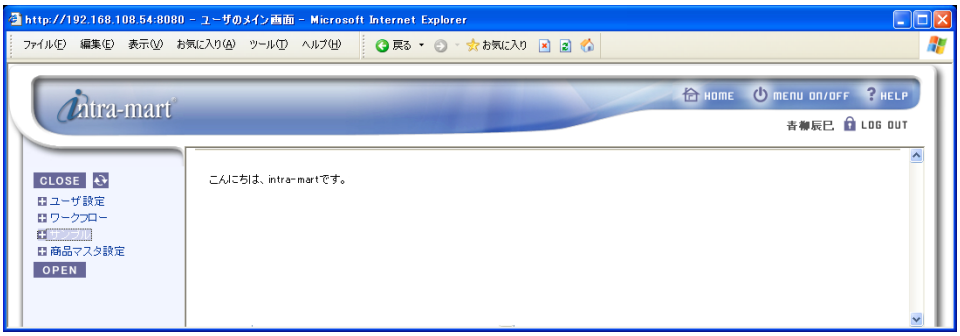
Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3. サーバサイド JavaScript より呼出された Java ファイルアクセスクラス（hello.js の（1）～（4））が外部テキストファイルの内容を読込みサーバサイド JavaScript へ結果を戻す。

＜hello.js（ファンクション・コンテナ）＞

```
var nameValue = ″ ″ ;
  //init 関数の定義
 function init(request) {
  //Java クラス FileInputStream を JavaScript オブジェクトとして生成
  var javaObjFileIn = new java.io.FileInputStream( ″c:¥¥hello.dat″ );          (1)

  //Java クラス DataInputStream を JavaScript オブジェクトとして生成
  var javaObjDataIn = new java.io.DataInputStream( javaObjFileIn );          (2)

  //ファイルを1レコード読み込み
  var javaObjString = javaObjDataIn.readLine( );                          (3)

  //ファイルクローズ
  javaObjDataIn.close( );                                              (4)

  //プレゼンテーション・ページへ値を引き渡し
  nameValue = Unicode.from(javaObjString);
}
```

4. サーバサイド JavaScript は Java ファイルアクセスクラスより取得した結果をプレゼンテーション・ページ（HTML）の<IMART>タグを結果に置き換えて出力する。



<hello.html の実行画面>

■ 解説

◆ コード変換 API

Unicode.*

intra-mart の実行環境は Unicode であるため、ファイルに保存されている内容などローカル文字体系の文字列を直接扱うことはできません。このような場合には、intra-mart に用意されているローカル文字体系の文字列を Unicode に変換する API を利用して変換する必要があります。詳細については「API リスト」を参照してください。

3.6.1.1 標準 JavaClasss 連携時の問題点

サーバサイド JavaScript と標準 JavaClass を連携する場合の問題点としては、標準 JavaClass 側で発生する例外を受けとる手段が存在していないという問題があります。

この問題はサーバサイド JavaScript で標準 JavaClass を直接使用する以上回避することはできないので注意が必要です。しかしまったく回避策がないわけではありません。以降に紹介する自作 JavaClass を作成し、インスタンス変数、メソッドなどを用意して例外が発生したか確認することは可能です。

3.6.1.2 自作 JavaClass との連携方法

自作 JavaClass との連携を行う場合、JavaClass ファイルの配置と自作 JavaClass 側、サーバサイド JavaScript 側の双方に特別な記述方法が必要となります。

1. JavaClass ファイルの配置
   自作 Java Class ファイルあるいは jar ファイルを、既定のフォルダに配置することで、intra-mart WebPlatform/AppFramework 上で動作させることができます。それぞれのファイルを以下通りに配置してください。
   - Class ファイル：doc/imart/WEB-INF/classes 配下
   - jar ファイル：doc/imart/WEB-INF/lib 配下
2. 自作 JavaClass 側の記述方法
   自作 JavaClass を作成する場合、次の点に注意して作成します。

```
package として作成する。
Class ファイルはクラスパスが通っている場所に配置する。
```

3. サーバサイド JavaScript 側の記述方法
   サーバサイド JavaScript を作成する場合は、自作 JavaClass を JavaScript オブジェクトとして生成する場合、Java の package 名の前に必ず「Packages」句を記述します。

```
var javaObj = new Packages.orgclass.myclass( );
```

以上のようにいくつかの点に注意するだけで簡単にサーバサイド JavaScript と自作 JavaClass と連携を行うことができるようになります。サーバサイド JavaScript は、Java プログラムなどのアプリケーション部品を呼び出すコンテナとしても十分に機能します。
このようにサーバサイド JavaScript をベースとした、より高度なアプリケーション開発が可能となります。
以下に自作 JavaClass を使用した記述例「HelloWorld」のリストを示します。

<hello.js（サーバサイド JavaScript ソース）>

```
var nameValue = " ";

//init 関数の定義
function init( requesst ) {

  //Java クラス hello を JavaScript オブジェクトとして生成
  var javaObjHello = new Packages.intramart.imartjava.hello();

  // helloClass の getHellostr メソッドによりファイルの１レコードを読み込む
  var javaObjString = javaObjHello.getHellostr( "c:¥¥hello.dat" );

  //Java クラスのエラー確認用インスタンス変数の値を読む
  var javaObjError = javaObjHello.errstr;

  //Java クラスエラーの判定
  if(javaObjError.substring( 0,2 ) == "ER") {

      //エラー時
      //プレゼンテーション・ページへ値を引き渡すオブジェクトへエラー内容を渡す。
      nameValue = Unicode.form(javaObjError);
  } else {

      //正常時
      //プレゼンテーション・ページへ値を引き渡すオブジェクトへ読込み内容を渡す。
      nameValue = Unicode.from(javaObjString);
  }
}
```

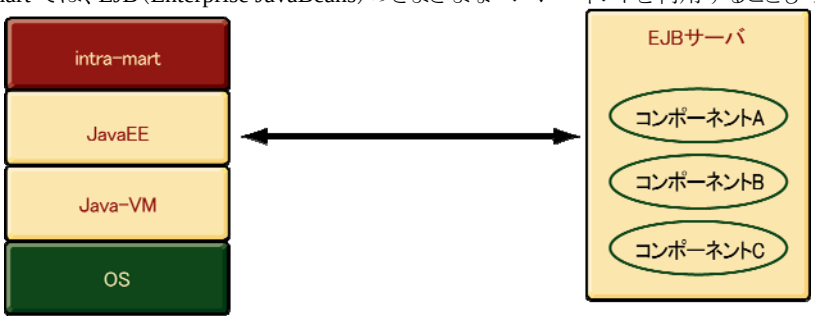

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

<hello.java（自作 JavaClass ソース）>

```
//パッケージ定義
package intramart.imartjava;

//クラスインポート
import java.lang.*;
import java.io.*;
import java.util.*;
//クラス定義
class hello {

	//例外などの ERROR 時確認インスタンス変数
	public String errstr;

	//コンストラクタ
	public hello( ) {
	}

	//ファイル読込みメソッド
	public String getHellostr( String fnamestr ) {

		//リターンする String 変数のインスタンス生成
		String readstr = new String();

		//インスタンス変数初期化
		errstr = "OK";
		try {
			//FileInputStream のインスタンス生成
			FileInputStream fs = new FileInputStream( fnamestr );

			//FileInputStream のインスタンス生成
			DataInputStream ds = new DataInputStream( fs );

			//ファイルを1レコード読み込み
			readstr = ds.readLine( );

			//ファイルの内容が null か確認
			if(readstr == null) {
				//インスタンス変数に ERROR をセット
				errstr = "ERROR1";
			}
			//ファイルをクローズ
			ds.close( );
		}
		//例外処理
		catch(IOException e) {
			//インスタンス変数に ERROR をセット
			errstr = "ERROR2";
		}
		//  読み取り内容をリターン
		return readstr;
	}
}
```

3.7 EJBとの連携

intra-mart では、EJB（Enterprise JavaBeans）のさまざまなコンポーネントを利用することも可能です。



<JavaEE を組み込むことで EJB サーバと連携>

3.7.1 EJBコンポーネントの作成

EJB の規約に準拠してクラスを作成します。JavaScript と連携する部分に関しては、自作クラスの呼び出しに準拠するようにします。作成した EJB コンポーネントの EJB サーバへの登録およびネーミング設定に関しては、各 EJB サーバ製品のマニュアルを参照してください。

3.7.2 JavaScriptからの呼び出し

クラスパスを適切に設定して、Application Runtime を起動します。ファンクション・コンテナ内で以下のようにして、目的の EJB コンポーネントを呼び出します。

<（例）XXX という EJB コンポーネントを呼び出す場合>

```
var initial = new Packages.javax.naming.InitialContext();
var objref = initial.lookup("XXX");
var home = Packages.javax.rmi.PortableRemoteObject.narrow
(objref.java.lang.Class.forName("XXXHome"));
var interfaceXXX = home.create();
```

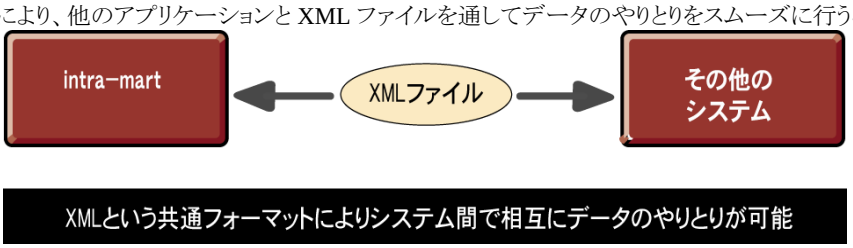
呼び出された EJB コンポーネントは、JavaScript の変数 interfaceXXX に格納されているので、あとは JAVA 呼び出しの要領で EJB コンポーネントの持つ各 API を実行できます。

※ 詳細に関しては、API リストの「JAVA クラスの利用」を参照してください。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.8 XML形式のデータを扱う

XML パーサーを利用することにより、XML 形式のデータを解析して、目的のデータを取り出すことができます。XML（Extensible Markup Language）は環境にとらわれない非常に柔軟性の高い汎用的な規約となっています。これにより、他のアプリケーションと XML ファイルを通してデータのやりとりをスムーズに行うことができます。



3.8.1 XMLパーサーとデータの取得

intra-mart の API として提供されている XML パーサーを利用すると、XML 形式のデータを解析して DOM（Document Object Model）ツリー形式に変換します。XML の各タグやその中に記述されているデータを DOM ツリーオブジェクトから取得する事ができます。

※ XML および DOM ツリーに関しては、W3C が規約を定めています。最新の情報に関しては W3C のホームページ等を参照してください。

※ XML パーサーに関しては、W3C および SAX のホームページ上で最新の技術情報が公開されています。

※ 詳細は、API リストの「アプリケーション共通モジュール」-「DOMXXX オブジェクト」、および、「XMLParser オブジェクト」を参照してください。

3.8.2 XML形式データの受信方法

ここでは、簡単な例として、以下の XML 形式データを受信するアプリケーションを作成します。

```
<?xml version='1.0' encoding='UTF-8'?>
<account>
    <user-id>ueda</user-id>
    <name>上田</name>
    <role>
        <role-id sample-attr="サンプル属性">level1</role-id>
    </role>
</account>
```

3.8.2.1 Request オブジェクトを使用した XML 形式データの受信方法

Request オブジェクトの getParameter()、および、getParameterValue()メソッドを利用することで、XML 形式データの値を参照することができます（これにより、Adobe Flash Player などのリッチクライアントから送信される XML 形式データを簡単に取り扱うことができます）。

Request#getParameter()の引数には、以下の形式に則ったパラメータ名を指定します。

- ◆ XML 形式データの各タグ名をセパレータ「/」で区切って指定する（ルートは「/」）
- ◆ 属性値を取得する際は、属性名の前に「@」を付与する

＜ファンクション・コンテナ(.js)の作成＞

```
function init(request){
        var userId          = request.getParameterValue("/account/user-id");
        var name            = request.getParameterValue("/account/name");
        var roleId          = request.getParameterValue("/account/role/role-id");
        var sampleAttr      = request.getParameterValue("/account/role/role-id/@sample-attr");

        Debug.browse(userId, name, roleId, sampleAttr);
}
```

※ この機能を利用するには以下の条件を満たしている必要があります。

- リクエストのメソッドが「POST」であること
- リクエストの Content-Type エンティティヘッダフィールドが「text/xml」であること
- リクエストのメッセージボディ部が構文解析可能な XML データであること

3.8.2.2 XMLParser オブジェクトを使用した XML 形式データの受信方法

＜ファンクション・コンテナ(.js)の作成＞

```
function init(request){

    //-----------------------------
    // メッセージボディを取得
    //-----------------------------
    var messageBody = request.getMessageBody("UTF-8");

    //-----------------------------
    // XML データの構文解析
    //-----------------------------
    var xmlParser = new XMLParser();
    var doc = xmlParser.parseString(messageBody);

    if(xmlParser.isError()){
        Debug.browse("エラーが発生しました。", xmlParser.getErrorMessage());
    }

    //-----------------------------
    // <user-id>, <name>, <role>ノード取得
    //-----------------------------
    var childNodes    = null;
    var accountNode = doc.getDocumentElement();
    var userIdNode    = null;
    var nameNode        = null;
    var roleNode        = null;

    childNodes = accountNode.getChildNodes();
    for(var i = 0 ; i < childNodes.length ;i++) {
        if(childNodes[i].getTagName() == "user-id") {
            userIdNode = childNodes[i];
        }
        else if(childNodes[i].getTagName() == "name") {
            nameNode = childNodes[i];
        }
        else if(childNodes[i].getTagName() == "role") {
            roleNode = childNodes[i];
        }
    }

    //-----------------------------
    // <role-id>ノード取得
    //-----------------------------
    var roleIdNode    = null;
    childNodes = roleNode.getChildNodes();
    for(var i = 0 ; i < childNodes.length ;i++) {
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

```
            if(childNodes[i].getTagName() == "role-id") {
                roleIdNode = childNodes[i];
            }
        }

        //-----------------------------
        // <role-id>ノードの属性取得
        //-----------------------------
        var roleIdAttr = roleIdNode.getAttribute("sample-attr");

        //-----------------------------
        //  各ノードの値を表示
        //-----------------------------
        Debug.browse(userIdNode.getChildNodes()[0].getValue(),
                     nameNode.getChildNodes()[0].getValue(),
                     roleIdNode.getChildNodes()[0].getValue(),
                     roleIdAttr);

}
```

3.8.3 XML形式データの送信方法

一般的な Web ブラウザは、受信したデータがどのような形式であるかを判定するために、レスポンスの Content-Type エンティティヘッダフィールドを利用します。サーバで作成した XML 形式のデータをクライアントに送信するには、レスポンスの Content-Type エンティティヘッダフィールドに"text/xml"を指定します。ここでは、簡単な例として、以下の XML 形式データをクライアントに送信するアプリケーションを作成します。

```
<?xml version='1.0' encoding='UTF-8'?>
<account>
    <user-id>ueda</user-id>
    <name>上田</name>
    <role>
        <role-id sample-attr="サンプル属性">level1</role-id>
    </role>
</account>
```

3.8.3.1 <IMART type="Content-Type">タグを使用した XML 形式データの送信方法

＜プレゼンテーション・ページ(.html)の作成＞

```
<IMART type="Content-Type" value="text/xml; charset=UTF-8"></IMART>
<?xml version='1.0' encoding='UTF-8'?>
<IMART type="string" value=xmlString></IMART>
```

＜ファンクション・コンテナ(.js)の作成＞

```
var xmlString = "";

function init(request){

    //-----------------------------
    // DOM ツリーを構築
    //-----------------------------
    var doc = new XMLDocument("<account/>");
    var accountNode = doc.getDocumentElement();

    //  エレメントを作成
    var userIdNode    = doc.createElement("user-id");
    var nameNode      = doc.createElement("name");
    var roleNode      = doc.createElement("role");
    var roleIdNode    = doc.createElement("role-id");

    //  テキストノードを作成
```

```
    var userIdText = doc.createTextNode("ueda");
    var nameText     = doc.createTextNode("上田");
    var roleIdText = doc.createTextNode("level1");

    // 属性を設定
    roleIdNode.setAttribute("sample-attr", "サンプル属性");

    // 子ノードを追加
    userIdNode.appendChild(userIdText);
    nameNode.appendChild(nameText);
    roleNode.appendChild(roleIdNode);
    roleIdNode.appendChild(roleIdText);

    accountNode.appendChild(userIdNode);
    accountNode.appendChild(nameNode);
    accountNode.appendChild(roleNode);

    //-----------------------------
    // XML の文字列をバインド
    //-----------------------------
    xmlString = doc.getXmlString();

}
```

3.8.3.2 HTTPResponse オブジェクトを使用した XML 形式データの送信方法

＜ファンクション・コンテナ(.js)の作成＞

```
function init(request){

    //-----------------------------
    // DOM ツリーを構築
    //-----------------------------
    var doc = new XMLDocument("<account/>");
    var accountNode = doc.getDocumentElement();

    // エレメントを作成
    var userIdNode    = doc.createElement("user-id");
    var nameNode       = doc.createElement("name");
    var roleNode       = doc.createElement("role");
    var roleIdNode    = doc.createElement("role-id");

    // テキストノードを作成
    var userIdText = doc.createTextNode("ueda");
    var nameText     = doc.createTextNode("上田");
    var roleIdText = doc.createTextNode("level1");

    // 属性を設定
    roleIdNode.setAttribute("sample-attr", "サンプル属性");

    // 子ノードを追加
    userIdNode.appendChild(userIdText);
    nameNode.appendChild(nameText);
    roleNode.appendChild(roleIdNode);
    roleIdNode.appendChild(roleIdText);

    accountNode.appendChild(userIdNode);
    accountNode.appendChild(nameNode);
    accountNode.appendChild(roleNode);

    //-----------------------------
    // XML の文字列をバインド
    //-----------------------------
    var encoding = "UTF-8";
    var xmlString = "<?xml version='1.0' encoding='" + encoding + "'?>" + doc.getXmlString();
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

```
    //-----------------------------
    // Content-Type を設定
    //-----------------------------
    var response = Web.getHTTPResponse();
    response.setContentType("text/xml; charset=" + encoding);

    //-----------------------------
    // データ送信
    //-----------------------------
    response.sendMessageBodyString(xmlString);

}
```

3.9 E4Xの利用方法

3.9.1 E4Xとは？

ECMAScript for XML (E4X) は、ネイティブ XML サポートを JavaScript に追加するプログラミング言語拡張です。E4X を利用することで、XML の階層構造を、JavaScript のプロパティのように「.(ドット)」で辿ることが出来たり、属性を「@(アットマーク)」で操作することが出来ます。

E4X は ECMA-357 標準で Ecma International によって標準化されています。
http://www.ecma-international.org/publications/standards/Ecma-357.htm
http://www.ne.jp/asahi/nanto/moon/specs/ecma-357.html(和訳)

3.9.2 XMLオブジェクトの作成

3.9.2.1 XML 構文から XML オブジェクトを作成する

E4X では、XML 構文をそのまま JavaScript コードに記述して、XML オブジェクトを生成することが可能です。JavaScript では、配列の初期化に [] を使用したり、オブジェクトの初期化に {} を使用することができるように、E4X では、XML の初期化に <> を使うことができます。

```
var xml = <root>
                <node attr="0">サンプル</node>
            </root>;
```

3.9.2.2 文字列から XML オブジェクトを作成する

E4X では、XML 形式の文字列から XML オブジェクトを生成することも可能です。

```
var src = "<root><node attr='0'>サンプル</node></root>";
var xml = new XML( src );
```

3.9.3 値の取得

E4X では、テキストノードの値や属性ノードの値を、JavaScript のプロパティのように「.(ドット)」で辿ることが出来たり、属性を「@(アットマーク)」で操作することが出来ます。

```
var xml = <root>
                <node attr="0">サンプル 0</node>
            </root>;

Debug.print(xml.node);              // 要素の値「サンプル 0」
Debug.print(xml.node.@attr);        // 属性の値「0」
Debug.print(xml.node["@attr"]); // このような形式でも属性 attr の値を取得することが可能です。
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.9.4 子ノードの取得

「children()」や「.*」を利用することで、子ノードを取得することが出来ます。

```
var xml = <root>
                <node1>AAAA</node1>
                <node1>BBBB</node1>
                <node2 num="0">CCCC</node2>
                <node2 num="1">
                    <node2Child>DDDD</node2Child>
                </node2>
            </root>;

 var children = xml.children();     // 「var children = xml.*;」と記述することも可能

 for(var prop in children){
     Debug.print("children[" + prop + "] = " + children[prop]);
 }

 // for each 文を利用して値を取得
 for each (var value in children){
     Debug.print("value = " + value);
 }
```

3.9.5 ノードの追加

以下のように、子ノード および 属性の追加が可能です。

```
var xml = <root>
                <node1>AAAA</node1>
            </root>;

 Debug.print("追加前：" + xml.toString());

 xml.addedNode        = "BBBB";   // 子ノードの追加
 xml.addedNode.@id = "CCCC";   // 属性の追加

 Debug.print("追加後：" + xml.toString());
```

3.9.6 ノードの削除

ノードの削除は、delete 演算子を利用します。

```
var xml = <root>
                <node1>AAAA</node1>
                <node1>BBBB</node1>
                <node2 num="0">CCCC</node2>
                <node2 num="1">
                    <node2Child>DDDD</node2Child>
                </node2>
            </root>;

 Debug.print("削除前：" + xml.toString());

 delete xml.node1[1];
 delete xml.node2[1].@num;

 Debug.print("削除後：" + xml.toString());
```

3.9.7 変数の挿入

E4X では、XML 文の一部を変数にすることもできます。XML 文中に中括弧で囲んで変数を挿入すると、オブジェクトの初期化時に対応する文字列と置き換えられます。

```
var attrName = "code";
 var tagName   = "name";

 var attrVal   = "0001";
 var content   = "商品1";

 var xml = <order>
                    <item {attrName}={attrVal}>
                         <{tagName}>{content}</{tagName}>
                    </item>
               </order>;

 Debug.print("============================");
 Debug.print(xml.toXMLString());
 Debug.print("============================");


 //「商品2」を追加
 attrVal = "0002";
 content = "商品2";
 xml.appendChild(
          <item {attrName}={attrVal}>
               <{tagName}>{content}</{tagName}>
          </item>
 );

 //「商品3」を追加
 attrVal = "0003";
 content = "商品3";
 xml.appendChild(
          <item {attrName}={attrVal}>
               <{tagName}>{content}</{tagName}>
          </item>
 );
 Debug.print("============================");
 Debug.print(xml.toXMLString());
 Debug.print("============================");


 //「商品2」を削除
 delete xml.item[1];

 Debug.print("============================");
 Debug.print(xml.toXMLString());
 Debug.print("============================");
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.10 JSSP-RPCについて

<IMART type="jsspRpc"> タグを利用すると、JavaScript で記述されたサーバサイドのロジックを、クライアントサイド JavaScript（以下 CSJS）からシームレスに呼び出すことが可能となります。

3.10.1 動作イメージ

サーバサイドに「**sample/test1**.js」が存在し、その JS ファイル内に「**testFunction**()」という関数が定義されている場合、以下の手順で CSJS からサーバサイド JS の関数を実行することができます。

1. HTML ファイル内に<IMART type="jsspRpc"> タグを以下のように記述します。

```
<IMART type="jsspRpc" name="serverLogic" page="sample/test1" >
```

2. CSJS 内に以下を記述することで、サーバサイドのロジックを実行します。

```
serverLogic.testFunction();
```

- サーバサイドの処理結果を非同期で受け取りたい場合は、属性 callback を指定します。サーバサイドの処理結果が 属性 callback で指定された CSJS 関数の引数に渡されます。詳しくは、API リストをご参照ください。

3.10.2 JSSP-RPC 通信エラーオブジェクトに関して

<IMART type="jsspRpc"> タグを利用したサーバサイドとの通信でエラーが発生した際、 そのエラー内容を格納したオブジェクトが伝達されます。

通信エラーが発生するのは以下の場合です。
レスポンスの HTTP ステータスコードが「200」以外の場合（サーバサイドで実行時エラーが発生した場合を含みます）
セッションタイムアウトが発生した場合
Debug.browse()を実行した場合

JSSP-RPC の通信方式（同期通信 または 非同期通信）によって、エラーオブジェクトの伝達方法が異なります。エラーオブジェクトの構成、および、伝達方法の詳細は、API リスト「jsspRpc」タグ の説明をご参照ください。

3.10.3 JSSP-RPC サンプルプログラム（同期通信）

サーバサイドの「jssp_rpc_test/sample1.js」に定義されている「getNow()」関数を実行後、「Now = （現在日付）」をアラート表示します。

3.10.3.1 クライアントサイドの HTML ソース

＜クライアントサイドの HTML ソース＞

```
<html>
    <head>
        <IMART type="jsspRpc"
                name="jsSample"
                page="jssp_rpc_test/sample1">
        </IMART>

        <script language="JavaScript">
            /**
             * 「jssp_rpc_test/sample1.js」の関数「getNow()」を実行します。
             */
            function execute(){
                try{
                    var result = jsSample.getNow("Now = ");
                    alert(result);
                }
                catch(ex){
                    alert(ex.message);
                    return;
                }
            }
        </script>
    <head>

    <body>
        <input type="button" value="実行（同期）" onclick="execute();">
    </body>
</html>
```

3.10.3.2 サーバサイドの JS ソース jssp_rpc_test/sample1.js

＜jssp_rpc_test/sample1.js＞

```
function getNow( args ){
    return args + (new Date()).toString();
}
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.10.4 JSSP-RPC サンプルプログラム（非同期通信）

サーバサイドの「jssp_rpc_test/sample2.js」に定義されている「getObject()」関数を実行し、
その結果オブジェクトをコールバック関数「callBackFunction」にて取得します。

3.10.4.1 クライアントサイドの HTML ソース

＜クライアントサイドの HTML ソース＞

```
<html>
    <head>
        <IMART type     = "jsspRpc"
               name     = "jsSample"
               page     = "jssp_rpc_test/sample2"
               callback = "callBackFunction">
        </IMART>

        <script language="JavaScript">
            /**
             * 「jssp_rpc_test/sample2.js」の関数「getObject()」を実行します。
             */
            function execute(){
                // 引数作成
                var obj = new Object();
                    obj.stringProp    = "value1";
                    obj.booleanProp   = true;
                    obj.numberProp    = -15;
                    obj.arrayProp     = new Array();
                    obj.arrayProp[0]  = "ary0";
                    obj.arrayProp[1]  = "ary1";
                    obj.arrayProp[2]  = "ary2";
                    obj.dateProp      = new Date();

                // 内容を確認
                var str  = "";
                    str += "コールバック関数を確認するために"                    + "¥n";
                    str += "サーバサイドで5秒間スリープします。"                  + "¥n";
                    str +=                                                       "¥n";
                    str += "実行前"                                              + "¥n";
                    str += "----------------------------------------"            + "¥n";
                    str += "obj.stringProp    = " + obj.stringProp               + "¥n";
                    str += "obj.booleanProp   = " + obj.booleanProp              + "¥n";
                    str += "obj.numberProp    = " + obj.numberProp               + "¥n";
                    str += "obj.arrayProp[0]  = " + obj.arrayProp[0]             + "¥n";
                    str += "obj.arrayProp[1]  = " + obj.arrayProp[1]             + "¥n";
                    str += "obj.arrayProp[2]  = " + obj.arrayProp[2]             + "¥n";
                    str += "obj.dateProp      = " + obj.dateProp                 + "¥n";
                    str += "----------------------------------------"            + "¥n";

                alert(str);

                // サーバロジック実行
                jsSample.getObject(obj);
            }

          /**
           * コールバック関数
            */
            function callBackFunction( result ){

                // 内容を確認
                var str  = "";
                    str += "実行後"                                              + "¥n";
```

```
                    str += "-----------------------------------------------" + "¥n";
                    str += "result.stringProp      = " + result.stringProp          + "¥n";
                    str += "result.booleanProp   = " + result.booleanProp          + "¥n";
                    str += "result.numberProp    = " + result.numberProp           + "¥n";
                    str += "result.arrayProp[0]   = " + result.arrayProp[0]        + "¥n";
                    str += "result.arrayProp[1]   = " + result.arrayProp[1]        + "¥n";
                    str += "result.arrayProp[2]   = " + result.arrayProp[2]        + "¥n";
                    str += "result.dateProp        = " + result.dateProp            + "¥n";
                    str += "-----------------------------------------------" + "¥n";

                alert(str);
            }

        </script>
    <head>

    <body>
        <input type="button" value="実行(非同期)" onclick="execute();">
    </body>
</html>
```

3.10.4.2 サーバサイドの JS ソース jssp_rpc_test/sample2.js

＜jssp_rpc_test/sample2.js＞

```
function getObject( args ){

    // 受け取ったオブジェクトの内容を表示
    for(var prop in args){
        if (args.hasOwnProperty(prop)) {
            Debug.print(prop + " : " + args[prop] + " [" + typeof args[prop] + "]")
        }
    }

    // コールバック関数を確認するために遅延処理を入れています。(5秒間)
    Client.sleep(5 * 1000);

    // 受け取ったオブジェクトの内容を加工
      args.stringProp        = args.stringProp    + " (modified !)";
      args.booleanProp      = false;
      args.numberProp      = args.numberProp + 10000;
      args.arrayProp[0]    = args.arrayProp[0] + " (modified !)";
      args.arrayProp[1]    = args.arrayProp[1] + " (modified !)";
      args.arrayProp[2]    = args.arrayProp[2] + " (modified !)";
      args.dateProp.setFullYear(2100);
      args.dateProp        = args.dateProp;

      // 結果を返却
      return args;
}
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.11 JavaScriptコンパイラ機能について

JavaScript コンパイラ機能は、JavaScript で記述されているファンクション・コンテナを Java クラスに変換（コンパイル）する機能で、次の2つのタイプがあります。

自動コンパイル	プログラム（ファンクション・コンテナ）実行時にアプリケーションサーバ(Application Runtime)が自動的にコンパイルします。以後、コンパイルされた Java クラスファイルを使って実行されます（サーバ稼動中にソースを変更しても反映されません）。 %Resource Service%/pages/src/source-config.xml の「resource-file/javascript/compiler」タグの enable 属性を ture にすることでこの機能が働きます。false にするとファンクション・コンテナはコンパイルされずに（インタプリタモード）動作します。（サーバ稼動中にソースを変更した場合、次のプログラム実行から変更が反映されます）
手動コンパイル	ファンクション・コンテナ作成後、JavaScriptコンパイラを利用し、予めJavaクラスに変換しておきます。（JavaScriptコンパイラについては「3.11.2 手動コンパイル」を参照してください）自動コンパイルよりもパフォーマンス向上が期待できます。運用時は、この方法で予め Java クラスファイルを作成して実行する方法を推奨します。

3.11.1 自動コンパイル

■ **source-config.xml ファイル**

source-config.xml ファイルは、source-config.xml ファイルが配置されているディレクトリ内(サブディレクトリを含む)のプログラムに対する設定ファイルです。

＜source-config.xml ファイルの設定例＞

```
<resource-file>
  <charset>Windows-31J</charset>
  <javascript>
      <compiler enable="true" />
      <!-- enable:true = Auto compiler to Java class -->
      <!-- enable:false = Interpreter -->

      <optimize level="0" />
      <!-- level:0 to 9 = Optimize level of Compile -->
  </javascript>
  <view>
      <compiler enable="true" />
      <!-- enable:true = Auto compiler -->
      <!-- enable:false = Interpreter -->

      <escapeXml enable="true" />
      <!-- enable:true = xml escape valid -->
      <!-- enable:false = xml escape invalid -->

      <escapeJs   enable="true" />
      <!-- enable:true = javascript escape valid -->
      <!-- enable:false = javascript escape invalid -->
  </view>

</resource-file>
```

source-config.xml ファイルでは以下の設定を行うことができます。

- resource-file/charset

ソースプログラムの文字エンコーディング名を指定します。デフォルト値は「サーバモジュールの文字コード」です。

- resource-file/javascript/compiler
  ファンクション・コンテナの自動コンパイルの有効・無効を設定します。
  この設定を有効(true)にすると、ファンクション・コンテナは実行時に Java クラスにコンパイルされて実行されます。(クラスファイルは、%Application Runtime%/work/jssp/ _functioncontainer ディレクトリに作成されます) 逆に、この設定を無効(false)にした場合は、ファンクション・コンテナは JavaScript インタプリタにより実行されます。デフォルト値は「false」です。
- resource-file/javascript/optimize
  ファンクション・コンテナを Java クラスにコンパイルする際の最適化レベルを設定します。デフォルト値は「0」です。
- resource-file/view/compiler
  プレゼンテーション・ページの自動コンパイルの有効・無効を設定します。
  この設定を有効(true)にすると、プレゼンテーション・ページがコンパイルされて実行されます。(クラスファイルは、%Application Runtime%/work/jssp/_presentationpage ディレクトリに作成されます) 逆に、この設定を無効にする場合は false を設定してください。デフォルト値は「false」です。
- resource-file/view/escapeXml
  IMART タグで出力される文字列の XML エスケープの有効、無効を設定します。
  この設定を有効(true)にすると、出力される文字列が XML エスケープされます。逆に、この設定を無効にすると、出力される文字列は XML エスケープされません。デフォルト値は「false」です。
  ecsapceXml 要素には、exclusion 属性と delimiter4exclusion 属性があります。exclusion 属性には、XML エスケープ処理の対象外とする文字列を指定してください。delimiter4exclusion 属性には、exclusion 要素に複数の文字列を指定した場合に利用した区切り文字を指定してください。
  詳細は、API リストの画面共通 IMART タグライブラリの string タグを参照してください。
- resource-file/view/escapeJs
  IMART タグで出力される文字列の JavaScript エスケープの有効、無効を設定します。
  この設定を有効(true)にすると、出力される文字列が JavaScript エスケープされます。逆に、この設定を無効にすると、出力される文字列は JavaScript エスケープされません。デフォルト値は「false」です。
  ecsapceXml 要素には、exclusion 属性と delimiter4exclusion 属性があります。exclusion 属性には、JavaScript エスケープ処理の対象外とする文字列を指定してください。delimiter4exclusion 属性には、exclusion 要素に複数の文字列を指定した場合に利用した区切り文字を指定してください。
  詳細は、API リストの画面共通 IMART タグライブラリの string タグを参照してください。

◆ ファイル単位での自動コンパイル設定方法

スクリプト開発モデルのプログラムは、プレゼンテーション・ページとファンクション・コンテナのペア単位で文字コードの指定や自動コンパイルの設定を行うことができます。「対象ファイルラベル名.properties」ファイルを作成し、以下のように設定することで動作します。
「対象ファイルラベル名.properties」ファイルがある場合、source-config.xml ファイルより優先され、「対象ファイルラベル名.properties」ファイルの設定内容が有効になります。

＜対象ファイルラベル名.properties＞

```
charset                      = プログラムの文字エンコーディング名
javascript.compile.enable    = ファンクション・コンテナの自動コンパイル設定
javascript.optimize.level    = ファンクション・コンテナを Java クラスにコンパイルする際の最適化レベル
view.compile.enable          = プレゼンテーション・ページの自動コンパイル設定
view.escapeXml.enable        = XML エスケープの有効・無効
view.escapeXml.exclusion= XML エスケープ処理の対象外とする文字列
view.escapeXml.delimiter4exclusion=「view.escapeXml.exclusion」に指定した文字列の区切り文字
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

```
view.escapeJs.enable       = JavaScript エスケープの有効・無効
view. escapeJs.exclusion  = JavaScript エスケープ処理の対象外とする文字列
view. escapeJs.delimiter4exclusion= 「view. escapeJs.exclusion」に指定した文字列の区切り文字
```

◆ 例えば、ファイルが文字コード「Windows-31J」で作成された sample.html と sample.js を、「ファンクション・コンテナの自動コンパイル機能を有効」、「プレゼンテーション・ページの自動コンパイル機能を無効」に設定する場合は、同一ディレクトリに sample.properties を作成し、以下の内容を記述します。

＜sample.properties ファイル＞

```
charset=Windows-31J
javascript.compile.enable=true
javascript.optimize.level=0
view.compile.enable=false
```

◆ **source-config.xml や properties ファイルの有効範囲**

下図のように source-config.xml ファイルと properties ファイルを配置した場合、各プログラムが参照する設定は以下の通りです。

* Aディレクトリ直下のプログラム：(3)の設定内容が有効になります。
* Bディレクトリ直下のプログラム：(3)の設定内容が有効になります。
* Cディレクトリ直下のプログラム：(2)の設定内容が有効になります。
* Dディレクトリ直下のプログラム：(2)の設定内容が有効になります。
* hoge.js/hoge.html ファイル　：(1)の設定内容が有効になります。

また、(2)の設定ファイルに記載がない項目は、(3)の項目の設定内容が有効になります。
（例えば、(2)に escapeXml の記載がない場合は、(3)の escapeXml の設定が有効になります。）

< source-config.xml の配置例>

```
Aディレクトリ
│
├──Bディレクトリ
│　　│
│　　└──Cディレクトリ
│　　　　│
│　　　　├────Dディレクトリ
│　　　　│
│　　　　├────hoge.html
│　　　　│
│　　　　├────hoge.js
│　　　　│
│　　　　├────hoge.properties・・・(1)
│　　　　│
│　　　　└────source-config.xml ファイル・・・(2)
│
└──source-config.xml ファイル・・・(3)
```

◆ **source-config.xml や properties ファイルの役割**

source-config.xml や properties ファイルは、ファイルやディレクトリ毎に自動コンパイルやエスケープの設定を可能にしています。以下のような場合に、ファイルやディレクトリ毎に設定すると便利です。

- ファイルやディレクトリ毎に異なる文字コードやコンパイルモードで開発を行いたい場合。
- 脆弱性対応（エスケープ対応）を機能別に順次リリースを行う場合に、特定のフォルダ、ファイルのみエスケープ対応を有効にすることが可能です。

3.11.2 手動コンパイル

3.11.2.1 形式

```
java -cp ./bin/intramart.jar jp.co.intra_mart.bin.js2class [ options ] [ sourcefiles ] [ sourcedirectories ]
( bin¥tools¥js2class [ options ] [ sourcefiles ] [ sourcedirectories ] )
```

- options
  コマンド行オプション
- sourcefiles
  コンパイルされる 1 つ以上のソースファイル (my_script.js など)
- sourcedirectories
  コンパイルされる 1 つ以上のソースファイルのあるディレクトリ

3.11.2.2 解説

jp.co.intra_mart.bin.js2class ツールは、**intramart.jar** アーカイブに含まれています。実行時には、**intramart.jar** からクラスがロードできるようにクラスパスを設定して下さい。(Service-Platform を Windows にインストールしている場合には、**bin/tools/** ディレクトリに **js2class.bat** というバッチファイルがインストールされています。このバッチファイルを利用することでクラスパスの設定等をしなくてもコンパイラを実行することができます。)
jp.co.intra_mart.bin.js2class ツールは、JavaScript プログラミング言語で記述されたファンクションコンテナを読み取り、バイトコードクラスファイル(JAVA のクラスファイル)にコンパイルします(*.js ファイルから *.class ファイルが作られます)。
ソースコードのファイル名を **jp.co.intra_mart.bin.js2class** に渡すには、ファイル名をコマンド行で直接指定します。ソースコードが複数ある場合は、各ファイル名またはソースコードの親ディレクトリ名をスペースで区切って指定します。
ソースコードのファイル名は .js 拡張子、クラスのファイル名は .class 拡張子を持たなければなりません。また、クラスファイル名はソースファイル名を元にして自動的に生成されます。このクラスファイル名は任意に変更することはできません。
内部関数定義は、追加のクラスファイルを生成します。これらのクラスファイルの名前は、_my_script_js$MyInnerFunction.class のように、内部クラス名と外部クラス名を組み合わせたものになります。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.11.2.3 オプション

-d sourcepath

ソースファイルを検索するソースコードパスを指定します。ソースファイルは、ソースコードパス **sourcepath** とソースファイル名の組合わせで検索します。

-d が指定されていない場合、**jp.co.intra_mart.bin.js2class** はカレントディレクトリからソースファイルを検索します。

-o directory

クラスファイルの出力先ディレクトリを設定します。クラスがパッケージの一部である場合、**jp.co.intra_mart.bin.js2class** は、必要に応じてディレクトリを作成し、パッケージ名を反映したサブディレクトリにクラスファイルを置きます。

-o が指定されていない場合、**jp.co.intra_mart.bin.js2class** はカレントディレクトリにクラスファイルを置きます。

-debug

デバッグ情報を生成します。行番号およびソースファイル情報などが生成されます。**-debug** オプションを指定することにより、実行時にエラーが発生した場合のデバッグが容易になります。

-opt level

実行時のコードを最適化します。最適化の程度は、0 から 9 までの整数値で指定することができます。数字が大きいほど最適化レベルは高くなりますが、その分コンパイル速度が低下し、デバッグの困難なプログラムが生成されることがあります。0 は最適化をせずにコンパイルします。**-opt** オプションを指定しなかった場合の標準値は 0 です。また **-opt** オプションは **-debug** と併用することはできません。

-charset charset

ソースファイルのエンコーディング名 (Shift_JIS/EUC-JP など) を指定します。**-charset** が指定されていない場合は、ご利用のシステムにおけるデフォルトの文字エンコーディングが使われます。 このオプションに指定できるエンコーディング名は、JAVA の仕様で定義されているエンコーディング名です。指定可能なエンコーディング名については、JAVA のドキュメントをご覧ください。

-h

ヘルプメッセージを出力します。他のオプション指定があっても、すべて無視されます。また -h オプションが指定された場合、コンパイルは行われません。

3.11.2.4 例(Windows の場合)

簡単なプログラムのコンパイル

サンプルソースファイル pages/src/sample/example/string/main.js をコンパイルする例を示します。(intra-mart のインストール時にサンプルをインストールしてください。)クラスファイルは、標準でクラスパスとなっている **doc/imart/WEB-INF/classes/** ディレクトリにコンパイルされるようにします。

このため、この例ではスタンドアロン環境の場合に限り、コンパイルした後にクラスファイルを移動したり実行時のクラスパスを設定する必要はありません(ネットワーク環境の場合は、作成されたクラスファイル(**_sample/_example/_string/_main_js.class** など)を Application Runtime の **doc/imart/WEB-INF/classes/** ディレクトリにコピーする必要があります)。

```
C:> dir
imart¥
C:> cd imart¥pages¥src
C:¥imart¥pages¥src> ..¥..¥bin¥tools¥js2class -o C:¥imart¥doc¥imart¥WEB-INF¥classes sample¥example¥string¥main.js
```

上記の例では、C:/imart/doc/imart/WEB-INF/classes ディレクトリに **_sample/_example/_string** ディレクトリが作成され、その中にクラスファイルが保存されます。この状態(ネットワーク構成の場合には、**_sample** ディレクトリを Application Runtime の **doc/imart/WEB-INF/classes** ディレクトリにコピー)でサーバを再起動すると、pages/src/sample/example/string/main.js は読み込まれなくなり、替わりにクラスファイルがロードされます(以後、ソースコードの変更は自動では反映されません)。

複数のソースファイルのコンパイル

次の例は、ソースファイル init.js とディレクトリ sample 内のすべてのソースファイルをコンパイルします。

```
C:> dir
imart¥
C:> cd imart¥pages¥src
C:¥imart¥pages¥src> ..¥..¥bin¥tools¥js2class -o C:¥imart¥doc¥imart¥WEB-INF¥classes init.js sample
```

3.11.2.5 その他

ソースファイルをコンパイルしておくことにより、システム全体のパフォーマンスを向上させることができます。ただし、ソースファイルを実行した場合とコンパイルされたクラスファイルを実行した場合では、システムにロードされる方式が異なるため実行時のスコープが変わってしまう場合があります。これにより実行時エラーが引き起こされる場合がありますので、ソースファイルをクラスファイル化した場合は、実行に問題がないことを必ず確認する必要があります。

コンパイルしたクラスファイルは、コンパイル時に指定した最適化レベルによりバイトコードの構造が異なります。特に最適化レベルが高い場合には、バイトコードが最適化されていることに起因して実行時エラーが発生する場合があります。その場合は、最適化レベルを低く設定するか、またはエラーとならないソースコードに書き換えて下さい。

コンパイルされたクラスファイルは、そのクラスファイル(ファンクションコンテナ)が実行されるサーバ(Application Runtime)環境においてクラスパスの設定されているディレクトリに保存して実行して下さい。クラスファイルは、JAVAの通常のクラスローダーによりロードされてVM上で実行されます。また、クラスファイルは一度実行されるとメモリ内にクラス情報がキャッシュされますので、クラスファイルを変更した場合はサーバの再起動が必要となります。

クラスファイル化したファンクションコンテナを使う場合は、Resource Service にソースファイルは必要ありません(クラスに該当するソースファイル(JavaScript ソースコード)は、削除してしまっても問題ありません)。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.11.3 実行時のファンクション・コンテナ検索手順

ファンクション・コンテナは、以下の手順に従って検索・実行されています。

1. クラスパスの中から手動コンパイルされたファンクションコンテナ（Java クラス）を検索
2. 自動コンパイル機能でコンパイルされた Java クラスを %Application Runtime%/work/jssp/_functioncontainer から検索
3. %Resource Service%/pages/src をルートディレクトリとしてソースファイルを検索
4. %Resource Service%/pages/product/src をルートディレクトリとしてソースファイルを検索
5. %Resource Service%/pages/platform/src をルートディレクトリとしてソースファイルを検索

上記プロセスにおいて、該当するファンクション・コンテナが見つかり次第、実行します。（自動コンパイル機能を利用している場合、手順 3～5 でソースファイルが見つかった場合に %Application Runtime%/work/jssp/_functioncontainer ディレクトリ以下に Java クラスファイルを作成しています）Java クラスファイルとソースファイルの混在による実行（例えば、一部のファンクション・コンテナのみコンパイルして、残りはインタプリタモードで実行）も可能です。

■ 解説

◆ **%Resource Service%の pages 配下のディレクトリ構成**

intra-mart Ver5.0 から%Resource Service%/pages 配下の構成が新しくなりました。開発者は、通常、%Resource Service%/pages/src にプログラムを格納します。

＜%Resource Service%/pages＞

```
│
├──platform
│    │
│    └──src   ← intra-mart WebPlatform/AppFramework のスクリプトプログラム格納用
│
├──product
│    │
│    └──src   ← intra-mart アプリケーション(イントラネット・スタートパックなど)のスクリプトプログラム格納用
│
│
│
└──src         ← 開発者が作成したスクリプトプログラム格納用
            （intra-mart WebPlatform のサンプルもこのディレクトリに格納されます）
```

3.11.4 仕様詳細

コンパイラによって生成される Java クラスの名称は、ファンクション・コンテナのファイルを元に決定されます。この時、ファンクション・コンテナのファイル名に Java クラス名として使用できない文字（下記の注意参照）が含まれていた場合、その文字をすべて"_"(アンダースコア)に置き換えます。元のファイル名に"_"文字が使われていて、他のクラス名と合致してしまった場合、エラーになってしまうことがあります。

※ クラス名として利用できない文字に関しては、Java の仕様に関するドキュメントを参照してください。

■ 自動コンパイル機能を利用している場合、ファンクション・コンテナ実行時に %Application Runtime%/work/jssp/_functioncontainer 内に Java クラスファイルが作成されます。プログラムを変更した場合にはサーバを再起動してください。サーバを再起動しても変更が反映されない場合、以下の手順で実行環境を初期化してください。

1. サーバ停止
2. work/jssp/を削除

3. サーバ起動

- JavaScript 関数名はファイル内でユニークである必要性があります。例えば関数内に宣言されている関数もこれに該当します。
- JavaScript コンパイル時に最適化機能を利用して作成したJava クラスファイルは、最適化せずにコンパイルした場合とバイトコードの構成が異なります。このため、最適化機能を利用した場合と利用しなかった場合で、エラー発生の有無や発生したエラーの内容が異なる場合があります。
- プログラム内容によりロードエラーになる場合があります。
- コンパイル後のコードサイズが大きすぎるとロードエラーになる場合があります。この場合は、各JavaScript 関数のコード量を少なくしてください(ファンクション・コンテナは JavaScript 関数ごとに Java クラスへコンパイルされます)。
- ファンクション・コンテナ呼出側のプログラムでプログラムパスの指定が曖昧な(大文字・小文字が完全に一致していない)場合、ロードエラーまたは実行時エラーになる場合があります(include()関数や <IMART> タグのリンクタグ等に対する page 属性など)。この場合は、呼出側で指定しているパスを正しいパスに修正してください。
- バッチ等のプログラム実行を設定するタイプのもので、設定しているパスが正しくない場合も同様の現象が発生します。この場合は、設定しているパスを正しいパスに再設定してください。
- プログラム内で使用されない変数が宣言されている場合、ロードエラーになることがあります。未使用変数の宣言はしないでください。

3.11.5 コンパイラとは直接関係ない部分の仕様

サイズの大きなオブジェクトをセッションに保存した際、パフォーマンスの劣化、および、サーバが異常終了する可能性があります。弊社では、64KB 以上のオブジェクトをセッションに保存することは推奨いたしません。スクリプト開発モデルの場合、対象となる API は以下のとおりです。

```
Client.set()
```

3.11.6 制約

3.11.6.1 ファイルサイズによる制約

プレゼンテーション・ページ(html)の<IMART>タグを除く静的なスクリプト部分のうち、連続した部分のサイズが 64［KB］を超える場合、実行時エラーになります。

3.11.6.2 プログラムの書き方による制約

以下の2つの条件を満たす場合、実行時エラーとなります。

- source-config.xml を以下のようにした設定した場合

```
resource-file/javascript/compiler タグの enable 属性を true にしている
resource-file/javascript/optimize タグの level 属性を 1 以上にしている
```

- 次のようなプログラムの書き方をした場合

```
init()関数および action 属性により実行される関数の両方から呼び出される共通関数を持っている。
init()関数から action 属性により実行される関数を呼び出している。
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

※ resource-file/javascript/optimize タグの level 属性を 1 以上にしている場合エラーとなるコード

＜test_page.html＞

```
<HTML>
<HEAD>
  <TITLE>Test Page</TITLE>
</HEAD>
<BODY bgcolor="WhiteSmoke">
  <CENTER>
      <HR>
          <!-- actionFunction 関数の呼び出し -->
          <IMART type="form" action="actionFunction">
              <INPUT type="submit">
          </IMART>
      <HR>
  </CENTER>
</BODY>
</HTML>
```

＜test_page.js＞

```
/**
* 初期化関数
* @param request Web リクエスト引数
*/
function init(request){
        //ここで actionFunction 関数を呼び出す
        //actionFunction 関数は test_page.html からも呼び出される
        actionFunction(null);
}
/**
* フォームの action 属性により呼び出される関数
* @param request Web リクエスト引数
*/
function actionFunction(request){
        Debug.print(viewTime().toString());
}
/**
* 共通関数
* @return 現在時刻を表す Date 型値
*/
function viewTime(){
        return new Date();
}
```

※ resource-file/javascript/optimize タグの level 属性を 1 以上にしている場合エラーとならないコード

＜test_page.html＞

```
<HTML>
<HEAD>
  <TITLE>Test Page</TITLE>
</HEAD>
<BODY bgcolor="WhiteSmoke">
  <CENTER>
      <HR>
      <!-- actionFunction 関数の呼び出し -->
      <IMART type="form" action="actionFunction">
          <INPUT type="submit">
      </IMART>
      <HR>
  </CENTER>
</BODY>
</HTML>
```

＜test_page.js＞

```
/**
* 初期化関数
* @param request Web リクエスト引数
*/
function init(request){
	//ここでは actionFunction 関数を呼び出さない
	//actionFunction 関数は test_page.html からのみ呼び出される
	Debug.print(viewTime().toString());
}
/**
* フォームの action 属性により呼び出される関数
* @param request Web リクエスト引数
*/
function actionFunction(request){
	Debug.print(viewTime().toString());
}
/**
* 共通関数
* @return 現在時刻を表す Date 型値
*/
function viewTime(){
	return new Date();
}
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.12 im-JavaEE Frameworkとの連携

im-JavaEE Framework はもとより、Servlet や JSP 等の画面からスクリプト開発モデルの画面へ遷移する場合、下記 API を利用します。

```
jp.co.intra_mart.jssp.net.URLBuilder
```

このクラスは、スクリプト開発モデルの画面を呼び出すための URL を作成するものです。リクエストのコンテキストを示す URL を生成する、以下のユーティリティクラスもあわせてご利用ください。

```
jp.co.intra_mart.common.aid.jsdk.utility.URLUtil
```

＜ソース例＞

```
// URLBuilder を生成
URLBuilder urlBuilder = new URLBuilder(request, response);

// リクエストのコンテキストを示す URL を生成
java.net.URL urlContext = URLUtil.getContextURL(request);

// HTTP セッションを維持したまま、
// 指定のスクリプト開発モデルの画面へリンクするための URL を取得
java.net.URL url = urlBuilder.createURLonSession(urlContext, "スクリプト開発モデルのページパス");

// この URL の文字列表現を構築
String nextPageURL = url.toExternalForm();
```

「スクリプト開発モデルのページパス」には、通常スクリプト開発モデルの実装において指定するページパスと同様のパス（%Resource Service%/pages/src からの相対）を指定してください。

3.13 グラフ描画モジュール

グラフの画像ファイルをサーバサイド作成して、ブラウザ画面上にグラフを表示します。

以下の 5 種類のグラフが利用できます。

- ◆ 折れ線グラフ
- ◆ 棒グラフ
- ◆ 円グラフ
- ◆ レーダーチャート
- ◆ ポートフォリオ

■ グラフ描画の設定

プレゼンテーション・ページ（HTML ファイル）にグラフ描画に関する<IMART>タグを記述します。用いられる<IMART>タグは以下の 5 種類です。

type 属性値	グラフの種類
lineGraph	折れ線グラフ
barGraph	棒グラフ
circleGraph	円グラフ
radarChart	レーダーチャート
portFolio	ポートフォリオ

折れ線グラフ描画の<IMART>タグの記述例は以下のようになります。

＜ソース例＞

```
<HTML>
  <HEAD>
      <TITLE>Line_Graph Sample</TITLE>
  </HEAD>
  <BODY>
      <IMART type="lineGraph"
          data=oData
          imageWidth="300"
          imageHeight="300"
          dataMin="-30"
          dataMax="60"
          scaleCount="20"
          alt="IM_LineGraph">
      </IMART>
  </BODY>
</HTML>
```

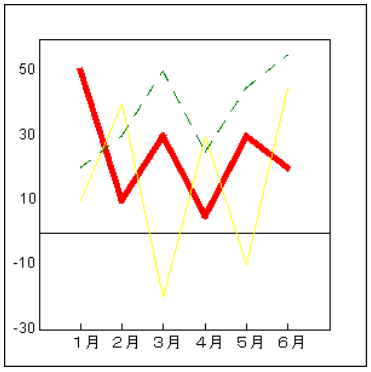
■ グラフの値となるオブジェクトの作成

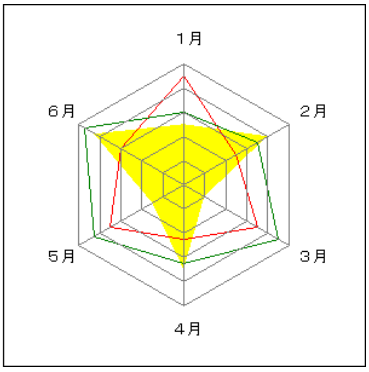
上記の<IMART type="lineGraph">タグの属性 data へのバインド変数はグラフのデータ値となり、以下のように、オブジェクトへ値をセットします。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

＜ソース例＞

```
//バインド変数宣言
var oData = new Object(); //  折れ線グラフ描画データ

//  ページの初期化関数
function init() {

        //  折れ線グラフ描画データオブジェクトの作成
        oData.aCaption = new Array("１月", "２月", "３月", "４月", "５月", "６月");
        oData.aData = new Array();
        oData.aData[0] = new Object();
        oData.aData[0].aData = new Array(50, 10, 30, 5, 30, 20);
        oData.aData[0].sColor = "red";
        oData.aData[0].nWidth = 5;

        oData.aData[1] = new Object();
        oData.aData[1].aData = new Array(10, 40, -20, 30, -10, 45);
        oData.aData[1].sColor = "yellow";

        oData.aData[2] = new Object();
        oData.aData[2].aData = new Array(20, 30, 50, 25, 45, 55);
        oData.aData[2].sColor = "green";
        oData.aData[2].nStyle = new Array(10,10);
}
```





<グラフ画面例>

3.14 アクセスコントローラモジュール

アクセスコントローラタグで挟まれている内容の、表示・非表示を制御することが可能です。

- ◆ アクセスコントローラタグで囲まれた領域がアクセスコントローラの制御範囲となります。
- ◆ アクセス権はロール、組織、役職、パブリックグループにより制御できます。
- ◆ アクセス権が存在しないユーザに対しては、アクセスコントローラ制御範囲の内容を非表示にします。

■ アクセスコントローラタグ

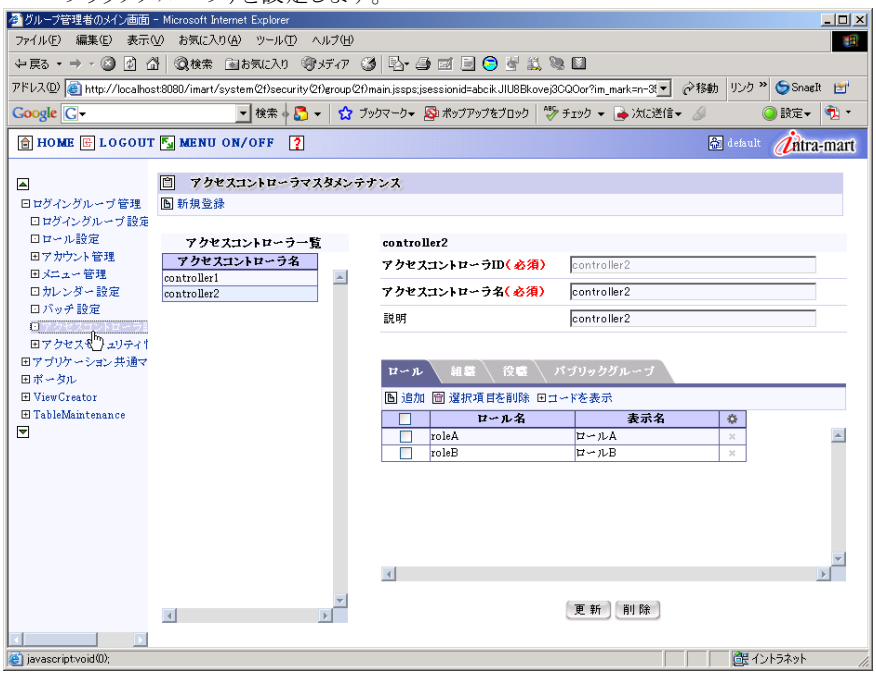
プレゼンテーション・ページにアクセスコントローラタグを記述することで、表示・非表示の制御を行います。

```
// アクセスコントローラ ID(controller1)に設定されたアクセス権で表示を制御します。
// controller1 のアクセス権情報が存在しない場合は、内容を表示しません。
<IMART type="accessCtrl" controller="controller1">
    アクセスコントローラタグ:controller1 で囲まれた内容です。
</IMART>

// アクセスコントローラ ID(controller2)に設定されたアクセス権で表示を制御します。
// controller2 のアクセス権情報が存在しない場合は、内容を表示します。
<IMART type="accessCtrl" controller="controller2" defaultShow="true">
   アクセスコントローラタグ:controller2 で囲まれた内容です。
</IMART>

// アクセスコントローラ ID(controller3)に設定されたアクセス権で表示を制御します。
// controller3 のアクセス権情報が存在しない場合は、内容を表示しません。
<IMART type="accessCtrl" controller="controller3" defaultShow="false">
    アクセスコントローラタグ:controller3 で囲まれた内容です。
</IMART>
```

■ アクセスコントローラのアクセス権設定

アクセスコントローラのアクセス権設定は、ログイングループ管理者の「アクセスコントローラ設定」画面で行います。

各アクセスコントローラ(アクセスコントローラ ID)に対して、表示させるための権限(ロール、組織、役職、パブリックグループ)を設定します。



Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.15 ショートカットアクセス機能

ショートカットアクセス機能は、初期アクセス URL にショートカットアクセス用の URL パラメータを指定することによって、ログイン後の画面を任意の画面に取り替えることができる機能です。
ショートカットアクセス機能を用いると、ログイン後の画面でトップバーおよびメニュー画面が存在し、そのメインエリアに任意のページを表示することが可能になります。
また、表示に関するセキュリティも設定することが可能です。
詳しくは、「アクセスセキュリティ仕様書」を参照してください。

ショートカットアクセス機能を用いた場合に利用できるセキュリティ機能は以下の通りです。

- ◆ 指定ページを表示できるユーザの指定。（複数設定可能）
- ◆ 指定ページを表示できる有効期間の指定。
- ◆ ログインの制御に関して以下の機能を選択できます。
  - ● ログイン画面からユーザ ID、パスワードを入力後、直接指定ページにアクセス。
  - ● ユーザ ID、パスワードを指定せずに、直接ページにアクセス可能。
    （表示許可ユーザを一名だけ指定した場合のみ利用できる機能です。）

■ ショートカットアクセス URL
ショートカットアクセス用の URL は、通常の初期アクセス URL にショートカット ID をパラメータとして追加した URL です。
http://<server>/<context-path>/<login-group>.portal?im_shortcut=<ショートカット ID>

＜記述例＞

```
http://localhost/imart/default.portal?im_shortcut=xazh03nbe43wd
```

■ ショートカット ID の作成
ショートカット ID は、表示するページの情報およびセキュリティの情報に紐づく ID となります。
ショートカット ID は、表示するページの情報およびセキュリティの情報を指定して API を用いて作成します。
メインエリアに、pages/src/sample/shortcut.js および shortcut.html を指定する場合のショートカット ID の作成手順を説明します。

＜ソース例＞

```
// ショートカットマネージャの作成
var manager = new ShortCutManager("default");

// ショートカット情報の作成
var shortCutInfo = new Object();

// 表示する URL
shortCutInfo.url = "sample/shortcut.jssp";

// 表示する URL に渡すパラメータの設定(任意指定)
shortCutInfo.urlParams = new Object();
shortCutInfo.urlParams["arg1"] = "value1";
shortCutInfo.urlParams["arg2"] = "value2";

// 表示許可を行うユーザ
shortCutInfo.allowUsers = new Array("guest","ueda");

// ログイン認証が必要かどうか？(認証必要)
shortCutInfo.isAuth = true;

// 以下の 2 通りの方法からどちらかを選択します。
// この情報の有効期限(作成時から10日間有効)
shortCutInfo.validEndDate = manager.addValidEndDate(10);
// この情報の有効期限(日付指定)
shortCutInfo.validEndDate = Module.date.get(9999,12,31);

// ショートカットID 作成
var shortCutId = manager.createShortCut(shortCutInfo);
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.16 外部ライブラリコール

3.16.1 概要

intra-mart フレームワーク では、外部ライブラリの呼び出しが可能です。
外部ライブラリの作成から実行までを解説します。

3.16.2 外部ライブラリコールのアーキテクチャ

外部ライブラリの呼び出しは、フレームワークから Java ラッパークラスを経由して、 呼び出しを行います。

外部ライブラリの呼び出しは以下の手順で行います。

1. ファンクション・コンテナの関数から Java ラッパークラスを生成します。
2. Java ラッパークラスの関数を呼び出します。
   （Java ラッパークラスの各関数が外部ライブラリの各関数と紐づいています。）
3. 対応した外部ライブラリの関数が実行されます。

3.16.3 Javaラッパークラスと外部ライブラリの作成

3.16.3.1 作成に必要なもの

Java ラッパークラスと外部ライブラリを作成するには、以下のものが必要です。

- JDK（Java ラッパークラスを作成するために必要です）
  - javac.exe (Java ラッパークラスをコンパイルします。)
  - javah.exe (Java ラッパークラスから C 言語用のヘッダファイルを作成します。)
  - jni.h（外部ライブラリ作成時に使用する標準ヘッダファイル）
- C コンパイラ（外部ライブラリを作成するために必要です）

3.16.3.2 フレームワークでの制約

フレームワークでは JavaScript から外部ライブラリを呼び出すために使用できる変数の型に制約を受けます。
以下に使用できる変数の型と対応する各言語の型を示します。

JavaScript の型	Java の型	C の型
String	String	jstring
Number	double	jdouble
Boolean	boolean	jboolean

3.16.3.3 作成手順

1. Java ラッパークラス（.java）を作成します。
2. javac.exe で Java ラッパークラス(.java)をコンパイルします。（.class の作成）
3. javah.exe で java ラッパークラス（.java）から、C 言語用のヘッダファイル(.h)を作成します。
4. 作成されたヘッダファイル(.h)を元に外部ライブラリファイル（.c）を作成します。
5. 外部ライブラリファイル(.c)をコンパイルし、外部ライブラリ（.dll）を作成します。

3.16.3.4 Java ラッパークラスの作成

1. 新規にクラスを作成します。
   (この時、パッケージ名を必ずつけてください。)
2. static エリア（初回 CLASS ロード時の処理）にロードするライブラリの名称を記述します。

   System.loadLibrary("ライブラリ名");
3. ライブラリ名には、拡張子を除いたものを記述します。

4. public なメソッドを記述します。
   public native 戻り型 関数名（引数の型 引数名, [.….] ）;
   戻り値 および 引数の型は String , double , boolean のいずれかです。
   ここでは、メソッドの宣言だけで、手続きは書かないことに注意してください。

5. 作成したファイルをコンパイルします。

   javac.exe ファイル名
   作成した class ファイルを.jar ファイルにもまとめます。

[作成例]

```
package Hellow;
public class Hellow {
        static { //  初回 CLASS ロード時の処理
        System.loadLibrary("Hellow");   //  ロードするライブラリ(Hellow.dll)
   }

   public Hellow( ) { }   //  コンストラクタ

   //  引数の文字列を結合する関数
        public native String margeString(String a , String b);
        //  引数の数値を合算する関数
        public native double getPlus(double a , double b);
        //  引数の論理積を求める関数
   public native boolean getAnd(boolean a ,boolean b);
}
```

3.16.3.5 外部ライブラリの作成

1. Java ラッパークラスから C 言語用のヘッダファイルを作成します。
   javah.exe ファイル名

   [作成されるヘッダファイルのプロトタイプ]
   JNIEXPORT 戻り型 JNICALL 関数名 (JavaVM のポインタ,クラスのオブジェクト,各引数の型);

2. C 言語用のヘッダファイルを元に外部ライブラリファイルを作成します。

   各関数の手続きを記述します。

3. 外部ライブラリファイルをコンパイルして、(.dll)を作成します。

3.16.3.6 jstring(文字列)の扱い方

文字列は Java 上では、UniCode として扱われます。
C 言語で、利用するためには、UniCode → UTF-8 への変換が必要です。
変換方法は以下の通りです。

*env : JAVA-VM のインスタンス（関数の引数から渡されます）
a : 変換したい文字列(jstring 型) UniCode
*ca : 変換後の文字列(char * 型) UTF-8

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

const char *ca = (*env)->GetStringUTFChars(env,a,0);
UTF-8 に変換すると、C 言語で通常の文字列として扱えます。

Java へ値を返す時は、UTF-8 → UniCode への変換が必要です。
以下の例は、C 言語の文字列から、Java の文字列を作成する方法です。
*env : JAVA-VM のインスタンス（関数の引数から渡されます）
jstr : 作成した文字列（jstring 型） UniCode
buf : 変換元の文字列(char * 型) UTF-8

jstr = (*env)->NewStringUTF(env,buf);

3.16.3.7 「作成されたヘッダファイルの例」

```
#include
#ifndef _Included_Hellow_Hellow
#define _Included_Hellow_Hellow
#ifdef __cplusplus
extern "C" {
#endif

JNIEXPORT jstring JNICALL Java_Hellow_Hellow_margeString
  (JNIEnv *, jobject, jstring, jstring);

JNIEXPORT jdouble JNICALL Java_Hellow_Hellow_getPlus
  (JNIEnv *, jobject, jdouble, jdouble);

JNIEXPORT jboolean JNICALL Java_Hellow_Hellow_getAnd
  (JNIEnv *, jobject, jboolean, jboolean);

#ifdef __cplusplus
}
#endif
#endif
```

3.16.3.8 「外部ライブラリファイルの例」

```
#include "Hellow_Hellow.h"
#include
/* 引数の文字列を結合する関数 */
JNIEXPORT jstring JNICALL Java_Hellow_Hellow_margeString
        (JNIEnv *env, jobject jobj, jstring a, jstring b)
{
  char buf[256]; /* テンポラリ文字列 */
  jstring jstr;   /* 戻り値用*/
  /* 引数をユニコードから UTF-8 への変換 */
  const char *ca = (*env)->GetStringUTFChars(env,a,0);
  const char *cb = (*env)->GetStringUTFChars(env,b,0);

  strcpy(buf,ca); /* コピー*/
  strcat(buf,cb); /* 結合   */
  /* 戻り値の値を Java 用の String として作成する。*/
  jstr = (*env)->NewStringUTF(env,buf);
  return jstr;
}
/* 引数の数値を合算する関数 */
JNIEXPORT jdouble JNICALL Java_Hellow_Hellow_getPlus
        (JNIEnv *env, jobject jobj , jdouble a, jdouble b)
{
  jdouble c; /* 戻り値用*/
  c = a + b; /* 合算する */
  return (c);
}
```

```
/* 引数の論理積を求める関数 */
JNIEXPORT jboolean JNICALL Java_Hellow_Hellow_getAnd
        (JNIEnv *env, jobject jobj, jboolean a, jboolean b)
{
  jboolean c ; /* 戻り値用*/
  c = a & b ; /* 論理積を求める */
  return (c);
}
```

3.16.4 外部ライブラリコールの方法

3.16.4.1 外部ライブラリコールの準備

1. 作成した外部ライブラリ(DLL ファイル)を PATH の通ったディレクトリに保存します。

2. 作成した Java ラッパークラス(.jar ファイル)を Application Runtime 起動時のクラスパスに追加します。
   intra-mart Administrator の JAVA 起動オプション画面から jar ファイル名(フルパスで)追加します。

3. Application Runtime を起動します。

3.16.4.2 JavaScript からの呼び出し

1. Java ラッパークラスを生成します。
   objHellow = new Packages.パッケージ名.クラス名();

2. 生成されたクラスから関数を呼び出します。
   Answer = objHellow.関数名(引数);

3. これで、外部ライブラリ関数の呼び出しが可能になります。

3.16.4.3 「JavaScript の例」

```
function init(request){
  var a;
  var b;
  var objHellow;

  objHellow = new Packages.Hellow.Hellow(); // クラスのロード

  a = "こんにちは " ;
  b = "イントラマートです"
  sAnswer = objHellow.margeString(a,b);

  // オブジェクト型で帰ってくるので、文字列にする必要があります。
  sAnswer = sAnswer + "";

  a = 10;
  b = 20;
  nAnswer = objHellow.getPlus(a ,b);

  a = true;
  b = false;
  bAnswer = objHellow.getAnd(a ,b);
}
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.17 バッチ管理モジュール

3.17.1 プログラムの作成

Resource Service のインストール・ディレクトリ以下にファンクション・コンテナとしてプログラムファイルを作成します。

バッチ起動されるファンクション・コンテナ内には、必ず init() 関数を定義する必要があります。

(指定時間に該当のファンクション・コンテナをロードすると init() 関数が Application Runtime により自動実行されます。)

バッチ起動されるファンクション・コンテナの init() 関数には、下記のような構造を持つオブジェクト型の引数が渡されます。

引数オブジェクト		
├	group	ログイン・グループ名称
├	name	バッチ設定名称
├	year	バッチ起動設定年
├	month	バッチ起動設定月
├	date	バッチ起動設定日
├	day	バッチ起動設定曜日
├	hour	バッチ起動設定時間
├	minute	バッチ起動設定分
├	second	バッチ起動設定秒
├	WWW_HOST	バッチ設定を行った時のWebサーバホスト名
├	WWW_PORT	バッチ設定を行った時のWebサーバポート
├	WWW_PROTOCOL	バッチ設定を行った時のWebサーバプロトコル
└	WWW_LOCATION	バッチ設定を行った時のホームURL

※ バッチ用プログラムに関しては、API リストの動作概要【 Batch Program 篇 】を参照して下さい。

3.17.2 システム構成

バッチ機能を利用するためには、すべてのサービスをそれぞれインストールし、Schedule Service から Application Runtime に対して HTTP 接続できるように環境構築してください。

3.17.3 バッチの登録と設定

バッチプログラムの登録と起動時間の設定は、i ntra-mart 起動後の『システム設定』-『バッチ』メニュー内で行います。

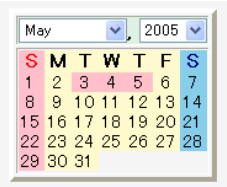
バッチ設定では、バッチ登録と運用可否設定があります。登録したバッチプログラムは、『有効』設定にしておかないと実行されません。

また、バッチサーバの実行状態を『運用中』にしなければバッチプログラムはロードされません。

設定画面の操作に関しては、マニュアルをご参照下さい。

3.18 カレンダーunit

カレンダーunit を組み込むと、カレンダーマスメンテナンス画面で設定したデータと連携したカレンダー画面を表示することができます。カレンダー画面を利用すると、会社の休日や営業日を考慮した日付の入力が行えます。



<カレンダーunit の例>

3.18.1 呼び出し方法

以下のキーワードをリンクすることによりカレンダーunit の画面を呼び出すことができます。

- @IM_CALENDAR_VIEW ： 標準の表示形式のカレンダー
- @IM_CALENDAR_VIEW_COMPACT ： 動作の軽い表示形式のカレンダー

＜ソース例＞

```
<IMART type="link"
  page="@IM_CALENDAR_VIEW"
      year="2008"
      month="5"
  link="sample/user1/calender_page">カレンダーへのリンク
  </IMART>
```

上記ソース例のようにリンクに対して指定する方法の他にも<IMART type="frame"> の src 属性や <IMART type="form"> または <IMART type="submit"> の page 属性に対しても同様に指定することができます。また、以下のオプション属性を指定することでカレンダー画面の動作を定義することができます。

- オプション属性

year	（必須） カレンダーの表示年
month	（必須） カレンダーの表示月
past	（任意） カレンダーの年選択コンボの表示量（過去） （現在表示年から過去 past 年間をコンボで選択可能）
future	（任意）カレンダーの年選択コンボの表示量（未来） （現在表示年から未来 past 年間をコンボで選択可能）
display	（任意）日がクリックされたときにページをコールするウィンドウ名 （または、フレーム名）
link	（任意）日がクリックされたときにコールされるページパス （Resource Service のプログラムディレクトリ （標準では、%Resource Service%/pages/src）からの相対パス）

3.18.2 カレンダーデータの受け取り方法

表示カレンダーの日がクリックされると自動的に link オプションに指定されているページをコールします。このページのファンクション・コンテナでユーザのクリック（選択）した情報を取得することができます。ユーザがクリックした日情報は request オブジェクトを介して取得することができます。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

＜ソース例＞

```
//calendar_page.js
Var oCalendar = new Object();
function init(request) {
oCalender.sGroup = request.group;  //  カレンダーID
oCalender.sYear = request.dtyear;   //  選択年の取得
oCalender.sMonth = request.dtmon; //  選択月の取得
oCalender.sDate = request.dtday;    //  選択日の取得
}
```

3.18.3 カレンダー拡張タグ と カレンダーモジュール

■ calendar タグ

<IMART type="calendar"> という拡張タグを使うことでカレンダー画面を自由に作成することができるようになります。

◆ 属性

year	作成するカレンダーの年
month	作成するカレンダーの月(1 - 12)
group	カレンダー作成時に利用するカレンダーID
format	使用するフォーマットファイルの指定
click_action	日リンクのクリック時に実行する関数名
weekCaption	曜日のキャプション（文字列の配列）。 日曜 - 土曜までのキャプションを配列で指定します。 未指定の場合は、デフォルト値{"S","M","T","W","T","F","S"} になります。

＜ソース例＞

```
<SCRIPT language="JavaScript">
function click_date(nDate, sName, bHoliday, nYear, nMonth, nDay) {
   //  クリックされた日付情報をもつ DATE 変数作成
     var dClick = new Date(nDate);
     //  画面表示文字列変数
     var sStr = dClick.toString();
     //  画面表示文字列の作成
     sStr += "  [ " + sName + " ]";
     sStr += "  [ " + bHoliday + " ]";
     sStr += "  [ " + nYear + "/" + nMonth + "/" + nDay + " ]";
     //  画面に表示して引数の確認
     window.alert(sStr);
}
</SCRIPT>
<IMART type="calendar"
year="1999"
month="5"
click_action="click_date">
</IMART>
```

■ CalendarManager

カレンダーマネージャオブジェクト。

カレンダーの参照、更新を行うマネージャオブジェクトです。

◆ 定数の概要

static String TYPE_CONFIG	名称(データタイプの定数)。
static String TYPE_HOLIDAY	休日(データタイプの定数)。
static String TYPE_NATIONAL_HOLDAY	祝祭日(データタイプの定数)。
static String TYPE_WEEKDAY	非休日(データタイプの定数)。

◆ メソッド一例

● カレンダーデータを新規に追加します。

```
Boolean addCalendarInfo (String calendarId ,CalendarInfo object)
```

● カレンダーを削除します。

```
Boolean deleteCalendar (String calendarId)
```

● カレンダーデータを削除します。

```
Boolean deleteCalendarInfo (String calendarId ,String dataId)
```

● すべてのカレンダーID を取得します。

```
Array getCalendarIds ()
```

● カレンダーデータを取得します。

```
CalendarInfo getCalendarInfo (String calendarId ,String dataId)
```

● カレンダーデータを更新します。

```
Boolean updateCalendarInfo (String calendarId ,CalendarInfo object)
```

全てのメソッドをご覧になる場合は、API を参照してください。

◆ パラメータ

calendarId String	カレンダーID
dataId String	カレンダーデータ ID
calendarInfoIds Object	カレンダーデータ ID の配列

■ CalenderInfo

カレンダデータオブジェクト。

カレンダデータオブジェクトは下記のプロパティを持つ Object 型のオブジェクトです。

◆ プロパティの概要

String color	背景色
Number day	日
Number dayWeek	曜日
String description	説明
String displayName	データ表示名
String id [必須]	データ ID
Number month	月
String name	[必須] データ名
String plugin	プラグイン ID
Number times	第n曜日(n:数値)
String type	[必須] データタイプ
Number validEndYear	有効終了年
Number validStartYear	有効開始年
Number week	第n週(n:数値)
Number year	年

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

＜オブジェクト作成方法＞

```
var obj = new Object();
obj.id = "";
obj.type = CalendarManager.TYPE_CONFIG;
obj.name = "";
obj.displayName = new Object();
obj.displayName["ja"] = "";
obj.displayName["en"] = "";
obj.description = "";
obj.year = 2005;
obj.month = 1;
obj.day = 1;
obj.dayWeek = -1;
obj.times = -1;
obj.week = -1;
obj.color = "#00ffff";
obj.plugin = "";
obj.validStartYear = 2000;
obj.validEndYear = 9999;
```

3.19 ツリー表示unit

ツリー表示 unitを組み込むと階層化されたデータをツリー表示することができ、階層構造の把握やメニューの選択が容易になります。

<ツリー表示ユニットの例>

＜ソース例＞

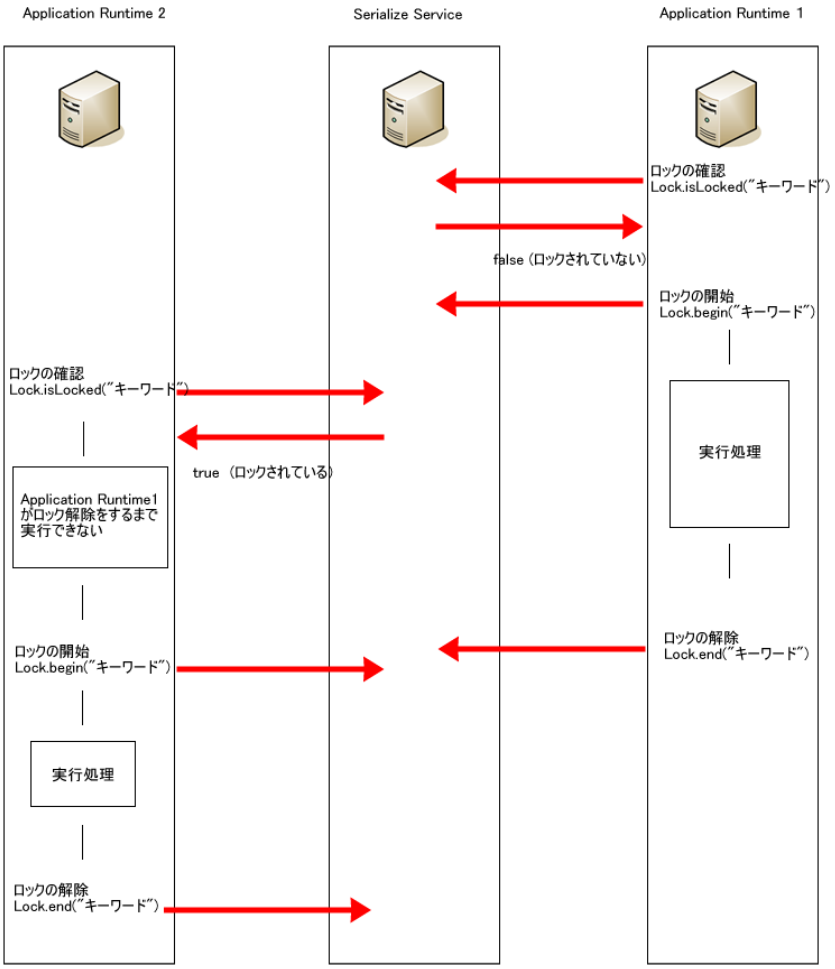
```
<!-- tree.html -->
<HTML>
  <HEAD>
    <SCRIPT language="JavaScript">
      function onClickHomeFunction(){
        window.alert("click home !");
      }
      function onClickFolderFunction(arg){
        window.alert("click folder: " + arg);
      }
      function onClickItemFunction(arg){
        window.alert("click item: " + arg);
      }
    </SCRIPT>
  </HEAD>
  <BODY>
    <IMART type="include"
          page="@TREE_VIEW"
          list=datas
          click_home="onClickHomeFunction"
          click_folder="onClickFolderFunction"
          click_item="onClickItemFunction"
          home_name="sample"
          home_detail="top"
          highlight="blue">
    </IMART>
  </BODY>
</HTML>
//tree.js
var datas = new Array();

function init(request){
  datas[0] = new Object();
  datas[0].name = "No.1";
  datas[0].detail = "item of No.1";
  datas[0].argument = "no1";
  datas[0].next = new Array();
  datas[0][0] = new Object();
  datas[0][0].name = "No.1-1";
  datas[0][0].detail = "item of No.1-1";
  datas[0][0].argument = "no1-1";
  datas[0][0].next = null;
  datas[1] = new Object();
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

```
    datas[1].name = "No.2";
    datas[1].detail = "item of No.2";
    datas[1].argument = "no2";
    datas[1].next = null;
}
```

3.20 アプリケーション・ロック機能

アプリケーション・ロック機能（処理のトランザクション）を実現します。Lock というAPIを利用することで、プログラムの直列処理を行うことができます。また、このAPIは、アプリケーションサーバが分散している場合においても、すべてのサーバで共通的にロックを掛けることができます（この機能は、Serialization Service を利用します）。



＜アプリケーション・ロックの直列処理＞

■ Lock

アプリケーション・ロック管理オブジェクト。

アプリケーション・ロックの制御を行うオブジェクトです。

◆ メソッド

● アプリケーション・ロックを開始します。

```
boolean begin (String name [,Number timeout])
```

● アプリケーション・ロックの開放します。

```
boolean end (String name)
```

● アプリケーション・ロックが開始されているかどうかを判別します。

```
boolean isLocked (String name)
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

＜ソース例＞

```
function insertStaff(){
        if(!Lock.isLocked("sampleApp")){
                Lock.begin("sampleApp",3);
                /*** sampleApp の実行処理 ***/
                DatabaseManager.beginTransaction();
                var sql ="insert into m_sample_stf ";
                        sql+= "values('stf011','シャイン 11','社員 11','staff-11')";
                var result = DatabaseManager.execute(sql);
                if(!result.error){
                        DatabaseManager.commit();
                }
                else{
                        DatabaseManager.rollback();
                }
                /*** 処理終了 ***/
        Lock.end("sampleApp");
   }
}
```

3.21 一意情報の取得機能

Identifier オブジェクトによって、Application Runtime が分散している場合においても、すべての Application Runtime でユニーク(一意)の情報を取得することができます(この機能は、Server Manager の管理情報を元に各 Application Runtime がシステム一意となるように制御します)。

- Identifier

  ユニークなIDを自動生成するオブジェクト。

  ユニークなIDを自動生成するAPIを持つオブジェクトです。

  - メソッド
    - ユニークな ID を作成します。

```
static String get()
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.22 製品のカスタマイズ

3.22.1 規定

intra-mart では、製品として提供されたプログラムを自由にカスタマイズして利用することができます。
カスタマイズが可能なプログラムは、オープン・ソースとして提供されているプログラムファイル(***.ini, *.html, *.js, *.properties, *.java** および , ***.jsp** など)すべてが対象となります。
製品のソース・コードに対してカスタマイズをした場合、カスタマイズをしたプログラムおよび動作に関連のあるプログラム群に関して、同パッケージの提供元は動作保証をいたしません。 また、カスタマイズをしたことにより発生した不具合に関しては、サポート対象外となります。

3.22.2 環境移行の手順

カスタマイズしたプログラムを別の環境へ移行（例えば開発機から本番運用機への移行）する場合、以下の手順で移行を行って下さい。

1. 移行先へ intra-mart をインストールします。
2. 移行先環境の intra-mart に対してライセンス登録を行います（アプリケーションを追加インストールしていてソースを展開している場合には、ソース展開も行います）。
3. 移行先環境の intra-mart を停止します。
4. カスタマイズした各プログラムソース(ini, html および js)を元環境から移行先環境へコピーします。
5. 移行先の intra-mart を起動します。

カスタマイズしたプログラムを移行する場合、**Resource Service** の **pages/** ディレクトリ内にある html および js 以外のファイルと **Permanent Data Service** の **treasure/** ディレクトリを上書きコピーしてしまわないように注意して下さい。万一、元環境から移行先環境に対してバイナリファイルの上書きコピーをしてしまいシステムが正常に動作しなくなった場合には、すべてのファイルを削除して移行先への intra-mart のインストールから再度行うようにして下さい。

3.22.3 注意事項

ソースが公開されているプログラムであっても、そのコード中に非公開のAPIを利用している場合があります。これら、非公開APIは予告なく仕様が変更されることがあるので注意が必要です。
製品のソースを直接カスタマイズした場合、カスタマイズをしたソースに関してはバージョンアップ対象外となります。パッチおよびバージョンアップ版のインストールの際にはソースが自動的に上書きされてしまうことがありますので、ご注意下さい。

3.23 アクセスセキュリティモジュールを利用しないで画面を構築する方法

3.23.1 概要

intra-mart では、アクセスセキュリティモジュールにとらわれない独自のアプリケーション作成を可能にするソリューションを標準機能として提供しています。

3.23.2 準備(インストール)

intra-mart をインストールガイドにしたがってインストールしてください。
分散システムを構築する場合には、すべてのサーバをインストールします。
インストールが完了したら、すべての Application Runtime の doc/imart/WEB-INF/web.xml にて 以下のフィルターマッピングの設定(<filter-mapping>)をすべて削除してください。
(intra-mart フレームワークの場合、この設定ファイル編集後に再デプロイを行ってください)

- RequestCharacterEncodingFilter
- ResponseCharacterEncodingFilter
- URLAccessFilter
- SessionFilter

3.23.3 作成(スクリプト開発モデル)

アプリケーション・プログラムを作成する場合、スクリプトファイルの動作仕様にしたがって、それぞれ目的にあったプログラムを記述していきます。
アプリケーションの動作概要と、スクリプトファイルの動作仕様に関しては、API リストの動作概要【 **Application Runtime 篇** 】および **Presentation Page** についてを参照して下さい。
作成したアプリケーションへは、以下のURLでアクセスできます（下記は標準インストールの場合の例です）。

```
http://<ホスト名>[:<ポート番号>]/imart/<作成したスクリプトのパス>.jssp
```

※<作成したスクリプトのパス>は、 Resource Service が管理しているリソースルートディレクトリからの相対形式で、拡張子(.html および .js)を省略したものを指定して下さい。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.23.4 注意事項

アクセスセキュリティモジュールを利用せずに画面を作成した場合、標準で提供されている一部の機能(API)が利用できません。

利用できないAPI(スクリプト開発モデル)

- アクセスセキュリティに関連した機能
  - AccessSecurityManager.*
  - AccountManager.*
  - LicenseManager.*
  - LoginGroupManager.*
  - RoleManager.*
  - UserManager.*
  - Module.client.*
- アプリケーション共通マスタに関連した機能
  - CategoryManager.*
  - CompanyManager.*
  - PrivateGroupManager.*
  - PublicGroupManager.*
  - Procedure.AppCommonUtil.*
- ワークフロー機能
  - AckApplicant.*
  - AckItem.*
  - Acknowledge.*
  - AckUtil.*
- その他
  - Module.alert.*
  - Module.external.*

利用できないAPI(JavaEE 開発モデル)

- アクセスセキュリティに関連した機能
  - jp.co.intra_mart.foundation.security.*
  - jp.co.intra_mart.foundation.acssecurity.*
- アプリケーション共通マスタに関連した機能
  - jp.co.intra_mart.foundation.datastore.*
  - jp.co.intra_mart.foundation.appcomn.*
- ワークフロー機能
  - jp.co.intra_mart.foundation.ackwkf.*

アクセスセキュリティモジュールを利用せずに画面を作成した場合、[install_directory]/conf/imart.xml 内の設定項目の中で有効に機能しないものがあります。

アクセスセキュリティモジュールを利用せずに画面を作成した場合、intra-mart の他のパッケージおよびアドオン・モジュールを追加インストールすることはできません。

3.24 検索ストリーミング機能

大量データ検索時の対応として、データベースから指定した件数ごとにレコードを取得して画面表示させるための関数（データベースフェッチメソッド）を用意しました。ワークフローの承認状況検索画面、申請状況検索画面では当メソッドを組み込んだ画面をサンプルとして提供しています。

■ DatabaseManager fetch

◆ メソッド

```
DatabaseManager.fetch(sql, stratRow, maxRow)
```

※ fetch メソッドにはこの他に、以下のパラメータを指定することができます。

◆ パラメータ

String sql	SQL(SELECT 文)
Number stratRow	取得するデータの開始レコード位置(1 以上)
Number length	取得するデータの最大レコード数(1 以上)
Array params	SQL 文に渡すパラメータの配列(DbParameter の配列)
String connectId	接続名
Boolean isGroup	true 引数の接続名をログイングループ ID としてログイングループのデータベースに接続します。 false 引数の接続名を利用してシステムデータベースに接続します。

■ ストアドプロシージャの使用手順

```
DatabaseManager.fetch(sql, stratRow, maxRow)
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

3.25 データベースのストアドプロシージャの呼び出し

■ DatabaseManager execStoredProc

DatabaseManager オブジェクトでは、データベースのストアドプロシージャを呼び出すこともできます。

◆ メソッド

```
DatabaseResult execStoredProc(String functionName)
```

※ execStredProc メソッドにはこの他に、以下のパラメータを指定することができます。

◆ パラメータ

String functionName	storedFunction 名
Array arg	引数オブジェクト(DbStoredProcArg の配列)
String connectId	接続名
Boolean isGroup	true 引数の接続名をログイングループ ID としてログイングループのデータベースに接続します。 false 引数の接続名を利用してシステムデータベースに接続します。

■ ストアドプロシージャの使用手順

1. intra-mart からデータベースのストアドプロシージャを呼び出すことができます。

```
CREATE PROCEDURE SelectMSampleStf AS SELECT * FROM m_sample_stf
```

2. DatabaseManager でストアドプロシージャを実行

```
var dbResult = DatabaseManager.execStoredProc("SelectMSampleStf");
```

3.26 国際化対応

MessageManager を利用することで、多言語のメッセージに対応することができます。
メッセージの多言語対応をするには、メッセージプロパティファイルが必要です。

■ メッセージプロパティファイルの設置場所

プロパティファイルは Java におけるプロパティファイルの仕様に準拠しています。
日本語のメッセージプロパティファイルは <ファイル名>_ja.properties とします。
各対応言語ごとのプロパティファイルを<%Server Manager%>/conf/message フォルダに設置してください。
設置が完了したら、MessageManager オブジェクトを使用して、各言語のメッセージの取得が可能になります。
※ プロパティファイル名は、<%Server Manager%>/conf/message フォルダ内でユニークな名前である必要があります。

◆ ntive2ascii ツール

日本語のメッセージプロパティファイルを作成する際、native2ascii ツールを使い、ネイティブコードを Unicode に変換する必要があります。

```
native2ascii inputfile outputfile
```

<例>

```
native2ascii sample_ja sample_ja.properties
```

※ 変換前のファイル名と変換後のファイル名が同じだと、正しく変換されません。
コマンド実行時はそれぞれ別のファイル名を指定してください。

■ MessageManager

メッセージマネージャオブジェクト。
メッセージ ID からメッセージ文字列を取得するオブジェクトです。

◆ メソッド

● 国際化されたメッセージを取得します。

```
static String getLocaleMessage (String locale ,String key ,String ....)
```

● メッセージを取得します。

```
static String getMessage (String key ,String ....)
```

◆ パラメータ

locale String	ロケール文字列
key String	メッセージID
...	String メッセージに挿入する文字列

※ locale の指定を省略した場合は、ログインユーザのデフォルトロケールが適用されます。
※ 取得メッセージに文字列等を挿入するには、プロパティファイル作成時に設定が必要です。

＜文字列挿入の設定方法＞
プロパティファイルの任意のメッセージで、挿入したい場所に配列を置きます。

＜(例)sample_massage.properties＞

```
SAMPLE.MESSAGE =  This is ({0}) message.
```

このメッセージに文字列を挿入して呼び出すには、次のような記述をします。

```
getMessage("SAMPLE.MESSAGE","sample");
```

(出力)
This is sample message.

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

※ 挿入する文字列が複数ある場合は、配列を増やします。

＜ソース例＞

```
MessageManager.getMessage("SETTING.GROUP.ADMIN_USER.MENU.TITLE");
```

■ Message タグ

<IMART type="message"> タグ。

タグの指定されている個所にメッセージ ID で指定された文字列を挿入します。

◆ 属性

id	メッセージ ID。
locale	ロケール。指定したロケールのメッセージを取得します。省略した場合は、ログインしているロケールが使用されます。
args	メッセージ引数。メッセージに置換用の定義がある場合、この引数の値に置換します。文字列の配列で指定します。

＜ソース例＞

```
<IMART type="message" id="message-id"></IMART>
<IMART type="message" id="message-id" args=aryArgs ></IMART>
<IMART type="message" id="message-id" locale="ja"></IMART>
<IMART type="message" id="message-id" locale="ja" args=aryArgs ></IMART>
```

4 デバッグ

4.1 デバッグ手順

開発者が作成した JavaScript に対して、Debug オブジェクトを用いてデバッグを行うことができます。デバッグを実行すると、デバッグメソッドで指定した部分のユーザ定義オブジェクトに関する名称、型、値、従属関係をデバッグ結果表示画面およびコンソール画面でチェックすることができます。

※ デバッグメソッドの詳細については、API リスト「アプリケーション共通モジュール」の「Debug.browse()」を参照してください。

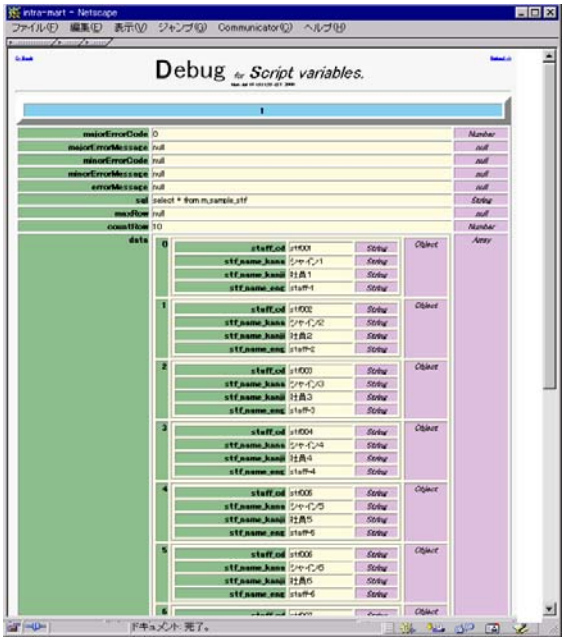
※ Debug.browse()メソッドを発行した時点で、デバッグページの表示が行われますので、それ以降のスクリプトは一切実行されません。

4.1.1 デバッグ例

以下にファンクション・コンテナにおけるデバッグの記述例とその実行結果であるデバッグ結果表示画面例を示します。

<デバッグ記述例>

```
// HTML へ渡す値を宣言します。
var nameVale;
var test;

// init 関数の定義
function init(){
	nameValue = Client.get( "nameValue" );		// HTML へ渡す値を設定します
	var newDate = new Date();
	var returnOfGetAge = procedure.getAge( newDate );
	test = returnOfGetAge;
	Debug.browse(newDate, returnOfGetAge);		//  変数のセット
}
```



<デバッグ結果表示画面>

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

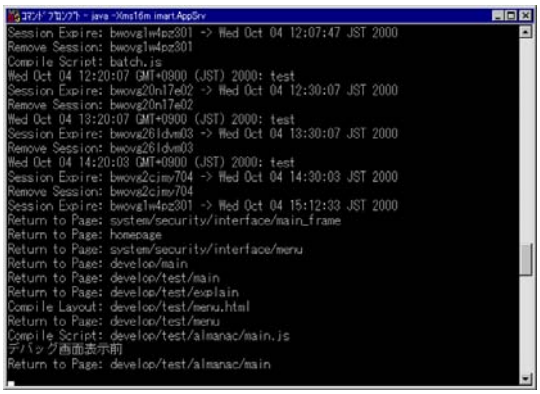
4.1.2 デバッグAPIの利用方法

ファンクション・コンテナをコーディング中に、変数の内容を確認したい場合が多々あります。
このような場合に利用するのがデバッグ API です。
intra-mart WebPlatform では、Debug クラスでこのような機能を提供しています。
実際にコーディング中に利用する場合、以下のようにして記述します。

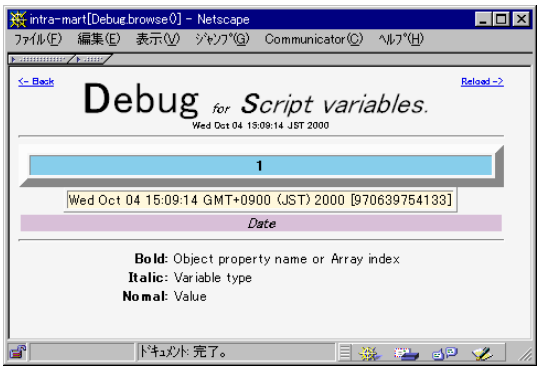
<sample.js>

```
//=========================================================
//          【入力】request: URL 引数取得オブジェクト
//          【返却】なし
  //          【概要】
  //=========================================================
function init(request){
var now = new Date();
  Debug.print("デバッグ画面表示前");        // コンソールに出力
  Debug.browse(now);                          // 画面に出力
      Debug.print("デバッグ画面表示後");       // コンソールに出力
  }
```

このサンプルソースでは、DOS コンソール画面に「デバッグ画面表示前」というメッセージを表示した後に、ブラウザ画面上に変数 now の内容（実行時の日時）を表示します。
9 行目で browse()API が実行されると、その時点でスクリプトの実行が中断されてブラウザ画面上にデバッグ画面が表示されます。よって、10 行目の print()API は実行されません。



<DOS コンソール実行画面（結果）-1>



<ブラウザ実行画面（結果）-2>

※ 詳細は API リストの「Debug」を参照してください。

■ 解説

◆ **Debug.print()**

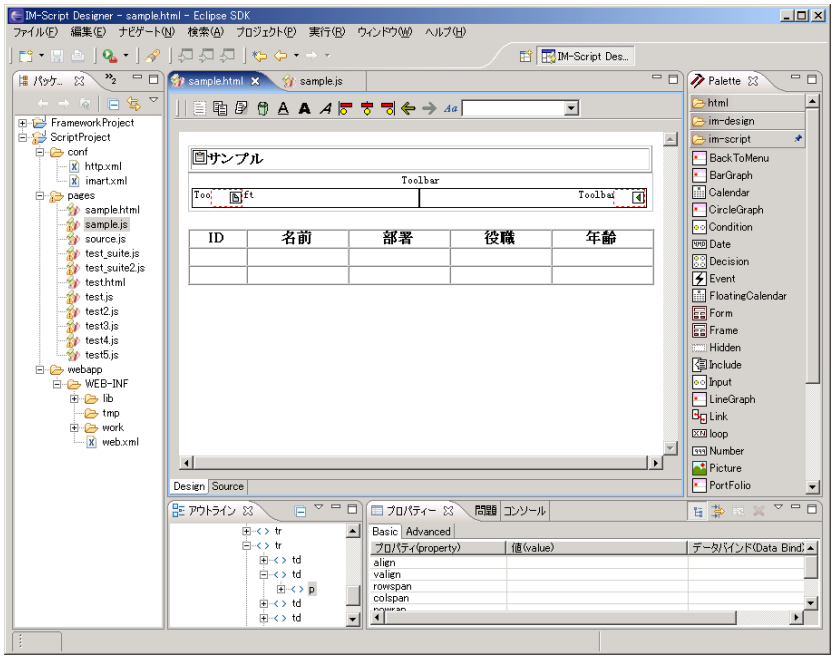
コンソールウィンドウに対してデバッグコードを出力することができるメソッドです。詳細は、「API リスト」を参照してください。

◆ **Debug.console()**

コンソールウィンドウに対してオブジェクトの内容を出力することができるメソッドです。出力される内容は JSON 形式の文字列です。詳細は、「API リスト」を参照してください

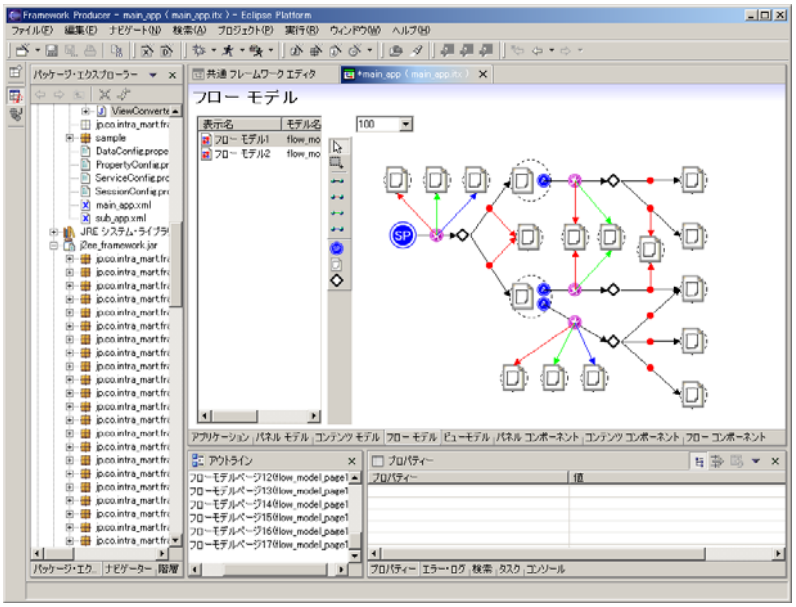
◆ **プログラム開発環境をサポートする eBuilder**

別売の「intra-mart eBuilder」を活用することにより、ユーザアプリケーションを効率よく開発していくことができます。「intra-mart eBuilder」には、オープンソースの統合開発環境である「eclipse」に対するプラグインとして利用できるプレゼンテーション・ページとファンクション・コンテナからなるスクリプト開発モデル用「intra-mart eBuilder Script Producer」と、JSP・Servlet からなる JavaEE 開発モデル用の「intra-mart eBuilder Framework Producer」の 2 種類が用意されています。



< intra-mart eBuilder Script Producer」>

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

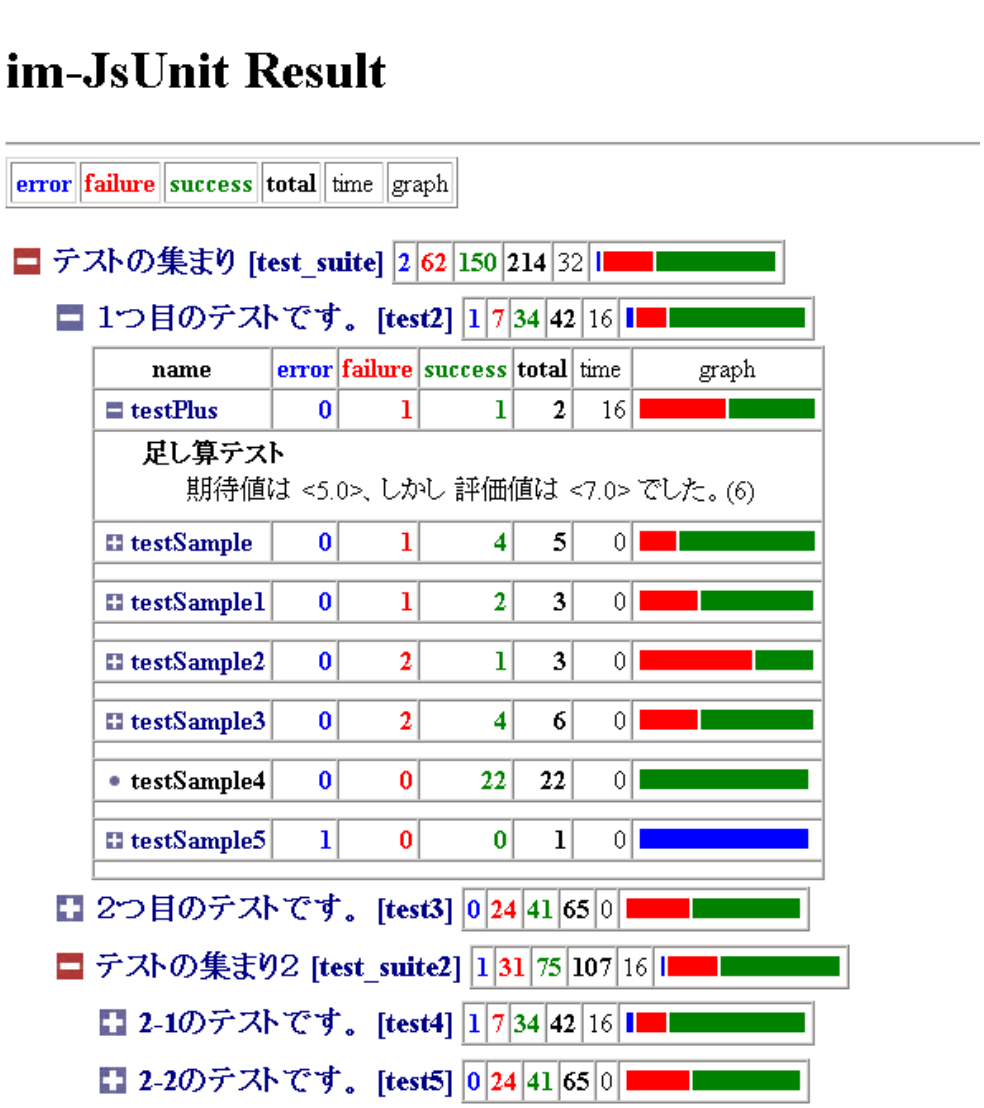


＜intra-mart eBuilder Framework Producer＞

4.2 単体テスト環境（im-JsUnit）

im-JsUnit は、スクリプト開発モデルにおける単体テスト環境を提供します。
スクリプトでテストケースを作成し、サーバ上でファンクション・コンテナの単体テストを実施します。

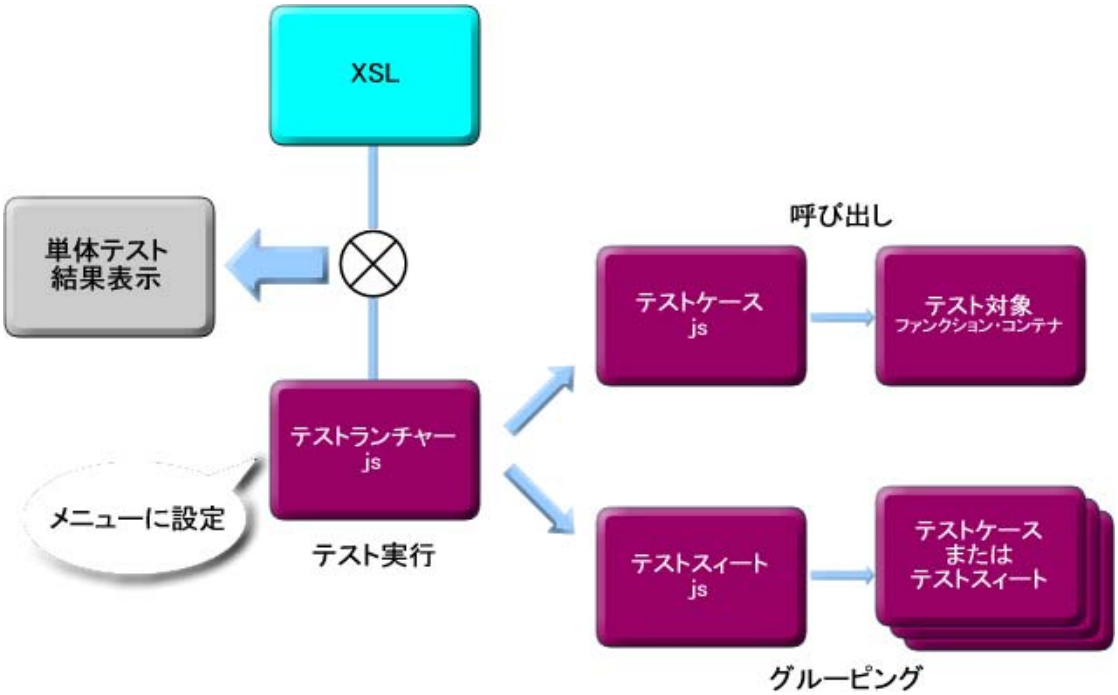
以下の図は、サーバで単体テストを実行した結果画面サンプルです。
テスト結果状況およびエラー状況が色覚的に分かりやすい表現になっています。



4.2.1 im-JsUnit概要

ユーザが作成したテストケースおよびテストスィートをテストランチャーを経由して実行します。
実行後、XSL でレイアウトを整形しテスト結果を画面上に表示します。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.



テスト対象	テスト対象は、テストの対象となるファンクション・コンテナです。 テスト対象に記述されている関数がテストの対象となります。
テストケース	テストケースは、テスト対象の関数または API に対してのテストケースを記述します。 JavaScript で記述します。拡張子は js です。
テストスィート	テストスィートは、複数のテストケースおよびテストスィートを1つのテストグループにするものです。 テストスィートを定義することで、複数のテストケースおよびテストスィートを一度に実行することが可能です。 JavaScript で記述します。拡張子は js です。
テストランチャー	テストランチャーは、指定された1つのテストケースおよびテストスィートを実行するファイルです。 ファンクション・コンテナ(js)のみで構成されます。 (プレゼンテーション・ページは不必要) 実行単位に作成し、メニューに登録します。 単体テストの実行は、このメニューから行います。
XSL	テストランチャーを実行した結果は XML 形式で出力されます。 XSL では、出力された XML を元に表示用レイアウトを作成しブラウザに表示します。 標準の XSL は xsl/jsunit/im_jsunit.xsl に定義されています。 テストランチャーのファンクション・コンテナ内で使用する XSL のファイルパスを指定します。

4.2.2 テストケースの実行順序

ここでは、テストケースに記述された各関数がどのような順序で実行されるかを解説します。

■ テストケースでの関数の種類

テストケース内で実行される関数の種類は以下の通りです。

testXXXXXX()	test から始まる関数を検索して、随時実行します。 各関数の実行する順序に決まりはありません。
setUp()	各 testXXXXXX()が実行される前に実行される関数です。 存在しない場合は実行されません。
tearDown()	各 testXXXXXX()が実行された後に実行される関数です。 存在しない場合は実行されません。
oneTimeSetUp()	テストケースファイルがロードされた直後に一度だけ実行されます。 存在しない場合は実行されません。
oneTimeTearDown()	テストケース内のすべての testXXXXXX()の実行が終わった後に一度だけ実行されます。 存在しない場合は実行されません。
defineTestSuite()	この関数が定義されていた場合、このファイルをテストスィートとして扱います。 その他の関数は無視されます。 テストスィートのファイルを作成する場合は、この関数だけを定義します。

■ 関数実行順序

テストケース内の関数の実行順序は以下の通りです。

テストケースファイルにテスト関数として testSample1() と testSample2() が記述されていた場合を例にすると、

① oneTimeSetUp()
② setUp()
③ testSample1()
④ tearDown()
⑤ setUp()
⑥ testSample2()
⑦ tearDown()
⑧ oneTimeTearDown()

の順序で実行されます。

4.2.3 テストケースの作成

■ テスト関数の作成

テストの内容は test から始まる関数内に定義します。

```
function testSample1() {
    テストの内容を記述します。
}
```

■ テスト対象ファイルのロード

テスト対象のファイルをロードするには以下の API を利用します。

テスト対象のパスは、拡張子を除いたものを指定します。

user/test/source.js がテスト対象の場合は user/test/source を指定します。

```
// テスト対象 user/test/source.js をオブジェクトとしてロードします。
var module = JsUnit.loadScriptModule("user/test/source");
```

テスト対象の内の関数を実行するには以下のように記述します。

```
// テスト対象 user/test/source.js をオブジェクトとしてロードします。
var module = JsUnit.loadScriptModule("user/test/source");

function testSample1() {
    // テスト対象の関数を呼び出します。
```


Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

```
    var result = module.calcPlus(1,2);
}
```

■ テストスィートの作成

テストスィートの作成は、defineTestSuite 関数を定義します。

テストスィートファイルには、defineTestSuite 関数のみを定義します。

defineTestSuite 関数内でテストスィートオブジェクトを作成し、グループ化したいテストケースおよびテストスィートファイルを追加します。

```
function defineTestSuite() {

    // テストスィートオブジェクトの作成
    var suite = new JsTestSuite("テストの集まり");

    // テストケース(テストスィート)を追加します。
    suite.addTest("1つ目のテストです。","test2");
    suite.addTest("2つ目のテストです。","test3");
    suite.addTest("3つ目のテストです。","test_suite2");

    // テストスィートオブジェクトを返却します。
    return suite;
}
```

4.2.3.1 評価関数の使用方法

評価関数は、テストの結果を評価するために利用する関数です。

この関数を用いることで、テスト結果として情報が収集されます。

■ 評価関数一覧

assert([comment] ,value)	評価値が True であることをチェックします。
assertEquals([comment] ,value1 ,value2)	評価値と期待値が同じであることをチェックします。
assertFalse([comment] ,value)	評価値が False であることをチェックします。
assertNaN([comment] ,value)	評価値が NaN であることをチェックします。
assertNotEquals([comment] ,value1, value2)	評価値と期待値が同じでないことをチェックします。
assertNotNaN([comment] , value)	評価値が NaN でないことをチェックします。
assertNotNull([comment] , value)	評価値が Null でないことをチェックします。
assertNotUndefined([comment] , value)	評価値が Undefined でないことをチェックします。
assertNull([comment] , value)	評価値が Null であることをチェックします。
assertTrue([comment] , value)	評価値が True であることをチェックします。
assertUndefined([comment] , value)	評価値が Undefined であることをチェックします。

＜記述例＞

```
// テスト対象 user/test/source.js をオブジェクトとしてロードします。
var module = JsUnit.loadScriptModule("user/test/source");

function testSample1() {
    // テスト対象の関数を呼び出します。
    var result = module.calcPlus(1,2);

    // 関数の結果が正しいかテストします。
    JsUnit.assertEquals(3,result);

    // 関数の結果が正しいかテストします。(コメント付)
    JsUnit.assertEquals("足し算のテスト(1 + 2)",3,result);
}
```

4.2.3.2 テストケース作成における注意点

テストケースを作成するにあたって、以下の点に注意してください。

- 画面遷移が発生する API を使用してはいけません。
  画面遷移が発生する API（Debug.browse(), redirect(), forward(), Module.alert.* など）を記述した場合、指定した画面に遷移するため正常に動作できません。
  別の関数に分けるなどして、テストを行ってください。

4.2.4 テストランチャーの作成

テストランチャーは、テストケースまたはテストスィートを実行するランチャーファイルです。

ファンクション・コンテナ（js）のみを作成します。

テストケースを実行する関数を利用してテストランチャーを記述します。

```
JsUnit.execute(テストケースパス,XSL パス);
```

テストケースパスは、実行するテストケースまたはテストスィートファイルを指定します。

user/test.js がテスト対象の場合は user/test を指定します。

XSL パスは、テスト結果をレイアウトするファイルを指定します。

通常は、xsl/jsunit/im_jsunit.xsl を指定してください。

この JsUnit.execute メソッドの戻り値は XML 形式の文字列となります。

テストケースファイル（test.js）を実行するには、テストランチャーファイルに以下のように記述します。

```
function init(request) {

    // テストを実行します。（結果は XML の文字列で返却されます。）
    var result = JsUnit.execute("user/test","xsl/jsunit/im_jsunit.xsl");

    // コンテントタイプの定義
    // 結果は、XML 形式で、エンコードは UTF-8 とする
        var response = Web.getHTTPResponse();
        Web.getHTTPResponse().setContentType("text/xml; charset=UTF-8");

        // データ送信
    response.sendMessageBodyString(result);
}
```

4.2.5 単体テストの実行

単体テストの実行手順を解説します。

1. テストケースファイルまたはテストスィートファイル（必要であれば）を作成します。
2. 実行したいテストケースファイルまたはテストスィートファイルのパスを記述したテストランチャーを作成します。
3. システム管理者またはグループ管理者のメニュー設定画面で②で作成したテストランチャーのパスを登録します。（ランチャーファイル名+.jssp で指定します。）
4. メニューよりランチャーを実行します。
5. テスト結果が画面に表示されます。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

5 サンプルプログラムの実行

5.1 サンプルのインストール

intra-mart インストール時に「サンプルをインストール」を選択すると、doc/imart または pages/src/sample 以下のディレクトリにサンプルがインストールされます。このサンプルはログイングループのメニューから実際にアクセスし、実行することができます。

スクリプト開発モデル	
サンプルプログラム・メニュー	ソースファイル・パス
［サンプル］-［スクリプト開発モデル］-[ショッピングカート]	pages/src/sample/item_sample/standard
［サンプル］-［スクリプト開発モデル］-[ショッピングカート[mskat]]	doc/imart/sample/item_sample/maskat/script
［サンプル］-［スクリプト開発モデル］-[グラフ]	pages/src/sample/graph/
［サンプル］-［スクリプト開発モデル］-[ファイル操作]	pages/src/sample/filer/
［サンプル］-［スクリプト開発モデル］-[メール送信]	pages/src/sample/mail/

※ ［サンプル］-［スクリプト開発モデル］- [ショッピングカート[maskat]]はマスカットを使用しているため、UTF-8 の環境のみ実行できます。マスカットを使用する場合は intra-mart をインストールする際の文字コードに UTF-8 を指定してください。

インストールされるサンプルは、プログラミングガイドで解説しているサンプルソースコードに比べ、実践的で上級者向けです。こちらのサンプルも合わせて活用することで、より深くスクリプト開発モデルのプログラミングを理解することが出来ます。

5.2 メニューのサンプル実行

右のメニューから[サンプル]-[スクリプト開発モデル]以下のサンプルメニューを選択します。

下図は[サンプル]-[スクリプト開発モデル]- [ショッピングカート]の例です。
[デモ]の画面では、サンプルを実行することができます。

<ショッピングカートの画面例 1>

[概要]の画面では、サンプルの目的や概要図等を確認することができます。
サンプルに関する詳細が載っていますので、参照してください。

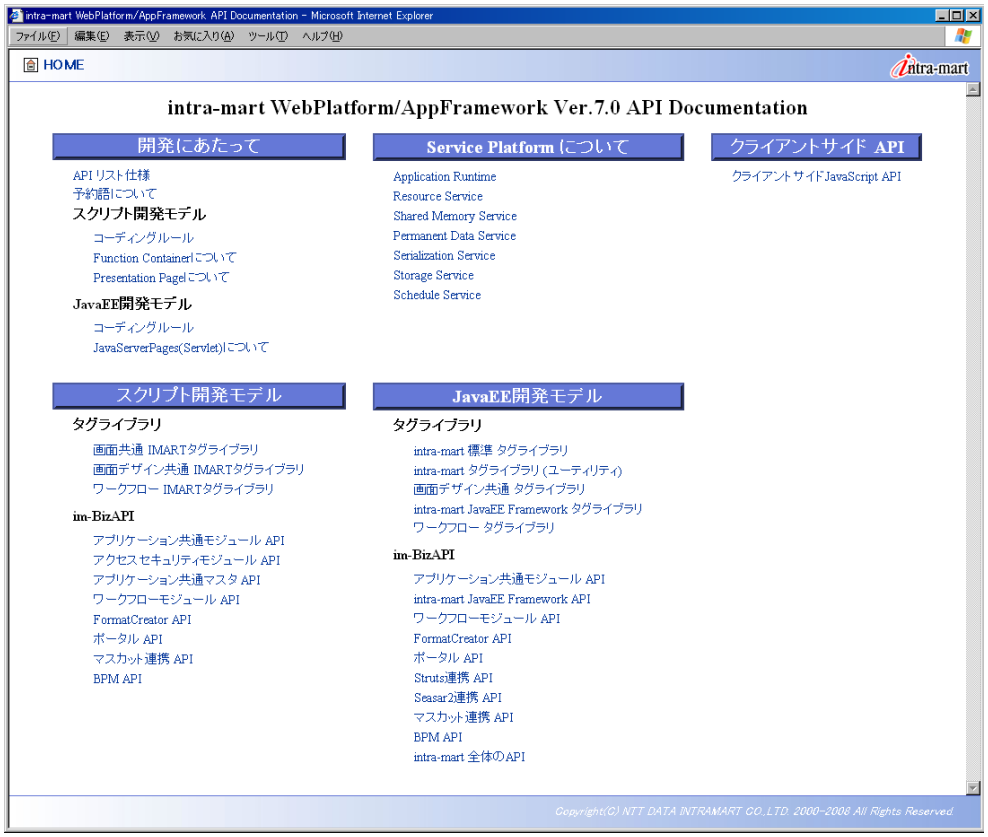
<ショッピングカートの画面例 2>

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

5.3 APIリスト

API リストは、ドキュメント CD の次の場所に格納されていますのでご利用ください。

HTML 形式 ： iwp_iaf/development/iwp_iaf_apilist_v70.zip



<API リスト>

6 Appendix

6.1 メッセージ設定

<%Server Manager%>/conf/message ディレクトリに、Java のプロパティファイル形式でメッセージを設定します。

※ メッセージ設定の詳細は「3.26 国際化対応」を参照してください。

ここで設定したメッセージを取得するには、MessageManager オブジェクトを利用します。 MessageManager には、メッセージ ID からメッセージ文字列を取得する getMessage()メソッドが用意されています。このメソッドは、ログインユーザのロケールを元にメッセージを取得します。

※ 詳細は「API リスト」の MessageManager オブジェクトの記述を参照してください。

6.2 予約語一覧

以下の用語は、intra-mart の中で予約語として使用されていますので、使用することができません。

- IMXXX (prefix が "IM(im)")
- _XXX（prefix が "_（アンダースコア）"）
- intra-mart API で使用されているクラス、およびグローバル関数名
  （intra-mart API では大文字を接頭辞として使用しています）
- JavaScript、Java での予約語

6.3 制限事項

アプリケーション作成時のファイル名および JavaScript 関数名には次のような制限があります。

6.3.1 ファイル名称

ファイル名称に、次の文字は使用できません。

```
¥ / : ; * ? ″ < > | & # [ ] ( ) { } (space) (tab)
(全角文字等日本語は使用できません)
```

※ ファイル名とは、プレゼンテーション・ページ（.html ファイル）とファンクション・コンテナ（.js ファイル）が対象です。データファイルはこれに含まれません。

6.3.2 ID、コード

intra-mart で提供している機能において、すべての ID、コード（ユーザ ID など）には、以下に示す文字を含めることはできません。

```
¥ / : , ; * ? ' ″ < > | & # + [ ] ( ) { } (space) (tab)
(全角文字等日本語は使用できません)
```

6.3.3 JavaScript関数

関数名称に、次の文字は使用できません。

```
* < > [ ]
(全角文字等日本語は使用できません)
(その他、JavaScript の仕様に依存します)
```

※ サーバ上で動作する関数に関しての制約です。クライアント上で動作する関数（HTML 内）に関してはこれに含まれません。また、関数だけでなく、その関数の登録名称やメソッド名にもこの制約は適応されます。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

6.4 スクリプト開発モデルの実行処理シーケンスについて

ブラウザから intra-mart に対してアクセスした場合の動作仕様を説明します。

ブラウザから HTTP リクエストにより スクリプト開発モデルのプログラムが実行される場合、ページを作成するために必要なプレゼンテーションページおよびファンクションコンテナを実行します。
ページプログラムは、プレゼンテーションページ(HTML)とファンクションコンテナ(JS)の2ファイルで1対となります。
2つのファイルは、ファイルラベル名で関連付けられます。それぞれのファイル(HTML および JS)は、必ず同じファイルラベル名で作成してください。
拡張子は、プレゼンテーションページの場合は .html で、ファンクションコンテナの場合は .js となります。
(どちらの拡張子も小文字のみとなります。大文字で記述した場合、正しく動作しなくなります。)
ファンクションコンテナが必要ない場合には、ファンクションコンテナファイルの省略（プレゼンテーションページファイルのみ）は可能ですが、プレゼンテーションページファイルの省略はできません。

ブラウザからリクエストがあった場合、以下のように処理が流れます。

1. ブラウザからのリクエスト受付
2. session.js 内 init() 関数実行
3. action 属性関数の実行
4. page 属性ファンクションコンテナ内 init() 関数の実行
5. page 属性プレゼンテーションページの実行
6. session.js 内 close() 関数実行
7. ページ返却

action 属性関数とは、intra-mart 連携用のリンク(<IMART type="link">)またはフォーム(<IMART type="form">)またはサブミット(<IMART type="submit">)からのリクエストであり、action 属性関数の実行指定がされていた場合にのみ処理されます(action 属性関数の実行指定がなかった場合には、この処理フェーズはスキップされます)。
action 属性関数には引数として、URL引数情報を持つオブジェクト(request)が渡されます。引数 request の詳細に関しては、[API リスト スクリプト開発モデル アプリケーション共通モジュール]を参照して下さい。
action 属性関数指定が複数同時に行われていた場合、以下の順位付けにしたがって実行対象となる関数を決定します。

link < form < submit

page 属性とは、intra-mart 連携用のリンク(<IMART type="link">)またはフォーム(<IMART type="form">)またはサブミット(<IMART type="submit">)からのリクエストであり、page 属性関数の実行指定がされていた場合にのみ処理されます。
page 属性指定がない場合は、リクエストをしてきたページを再実行します。

page 属性ファンクションコンテナ内 init() 関数には引数として、URL引数情報を持つオブジェクト(request)が渡されます。引数 request の詳細に関しては、[API リスト スクリプト開発モデル 画面共通モジュール] を参照して下さい。

プレゼンテーションページでは、<IMART> タグ部分が実行されます。
プレゼンテーションページ内のAPIについては、[API リストスクリプト開発モデル 画面共通モジュール] を参照して下さい。

各スクリプトの実行フェーズにおいて、何らかの要因でエラーが発生した場合、即座にスクリプト実行を中止して、ブラウザに対してはエラー画面を送信します。
エラーが発生した場合は、エラーの発生した実行フェーズおよび『session.js 内 close() 関数実行』および『ページ返却』を除く残りの実行フェーズはスキップされます。

session.js 内 close() 関数実行フェーズは、ページ処理に関する終了処理を行うことを目的とします。
session.js 内 close() 関数実行フェーズにおいて、Debug.browse() や forward() などのスクリプト実行を強制的に中断して処理を遷移させるAPIを実行した場合、正しく処理されない場合があります。

6.4.1 ファンクションコンテナの種類

ファンクションコンテナは、サーバで実行される JavaScript で記述された intra-mart のビジネスロジック・プログラム・ファイルです。(Client Side JavaScript は、これに含まれません)
ファンクションコンテナは、用途により4種に大別されます。

- 初期起動用
- Web ページ連携用
- バッチプログラム用
- 特殊プログラム

※ 共通関数登録用および独立型関数定義用などは、その性質の類似から『初期起動用』と同義と扱います。

6.4.1.1 初期起動用

初期起動用ファンクションコンテナは、サーバによりプログラムファイルがロードされると、ファイル内に記述された init() 関数が実行されます。
共通関数登録などの初期化処理は、init() 関数内に記述するようにしてください。
サーバの初期起動時には、ソースのルートディレクトリ（標準では、pages/src または pages/platform/src）にある init.js 内に定義された init() 関数が実行されます。

■ 実行シーケンス
1. init.js のロード
2. init.js 内の init() 関数の実行

6.4.1.2 プレゼンテーションページ連携用

プレゼンテーションページと連動するファンクションコンテナは、サーバにより実行される時に、引数を伴って呼び出されます。
HTTP リクエストにより実行された場合、URL引数を持つ request オブジェクトを引数として init() 関数が実行されます。
また、リンクやフォームの action 属性により任意の関数実行が指定されている場合、URL引数を持つ request オブジェクトを引数として指定の関数が実行されます。

HTTP リクエスト以外の方法(API forward() 等)により実行された場合、ファンクションコンテナ内の init() 関数は、その方法によって定義された引数を伴って実行されます。(詳細に関しては、各々のAPI仕様を参照してください)

プレゼンテーションページの実行が完了したあと、ファンクションコンテナ内に close() 関数が定義されていた場合、URL引数を持つ request オブジェクトを引数として close() 関数が実行されます。

Copyright 2000-2013 株式会社 NTT データ イントラマート All rights Reserved.

■ 実行シーケンス

1. action 属性指定関数の実行
2. init() 関数の実行
3. プレゼンテーションページの実行
4. close() 関数の実行

action 属性関数の実行指定がない場合は、action 属性関数の実行フェーズはスキップされます。
プレゼンテーションページ実行中は、プレゼンテーションページ側の要求に応じて、ファンクションコンテナ内の任意の関数が実行されることがあります。

6.4.1.3 バッチプログラム用

intra-mart Batch Server により、定められた時刻に実行されるファンクションコンテナが、これに該当します。
バッチ設定メンテナンスにより設定されている時刻になると、設定された任意のファンクションコンテナ内に定義された init() 関数が Application Runtime により実行されます。
バッチ起動されるファンクションコンテナの init() 関数には、下記のような構造を持つオブジェクト型の引数が渡されます。

```
引数 オブジェクト
├ group ログイン・グループ名称
├ name バッチ設定名称
├ year バッチ起動設定年
├ month バッチ起動設定月
├ date バッチ起動設定日
├ day バッチ起動設定曜日
├ hour バッチ起動設定時間
├ minute バッチ起動設定分
├ second バッチ起動設定秒
├ WWW_HOST バッチ設定を行った時のWebサーバホスト名
├ WWW_PORT バッチ設定を行った時のWebサーバポート
├ WWW_PROTOCOL バッチ設定を行った時のWebサーバプロトコル
└ WWW_LOCATION バッチ設定を行った時のホームURL
```

6.4.1.4 特殊プログラム

session.js は、Application Runtime が HTTP リクエストを受け付けた時に実行されるプログラムです。
Application Runtime が HTTP リクエストを受け付けると、session.js 内に定義された init() 関数が実行されます。
また、Application Runtime がプレゼンテーションページの作成を完了した後、HTTP レスポンスを返す前に close() 関数が実行されます。
session.js ファイルを変更した場合は、Application Runtime を再起動するまでシステムには反映されません。

6.4.1.5 注意事項

ファンクションコンテナ内に定義された init() 関数および close() 関数は、特別な意味を持ちます。
これらの関数を正規の目的以外に定義をすると、思わぬ誤動作の原因となります。

intra-mart WebPlatform/AppFramework Ver.7.0
スクリプト開発モデル プログラミングガイド

2013/10/18 第8版

Copyright 2000-2013 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/