



## intra-mart WebPlatform/ AppFramework

# プログラミングガイド スクリプト開発モデル編

Ver 6.1

❖ 変更履歴

変更年月日	変更内容
2007/07/31	初版
2007/08/31	第二版 『1.5 2つのWebアプリケーションモデル』を修正 『1.6 intra-martのアプリケーション開発概要』を修正 『2.3 データベースからデータを取得する』を修正
2009/03/27	第三版 『3.13.1 実行時のファンクション・コンテナ検索手順』を修正
2009/09/30	第四版 『3.13.3 コンパイラとは直接関係ない部分の仕様』を修正 『3.19.13 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 『3.4.1 グローバル関数としての登録・呼び出し』の誤字を修正

## Contents

第1章 イントロダクション	1
1.1 intra-martの起動と終了	2
1.1.1 intra-martへのログイン	2
1.1.2 intra-martからのログアウト	3
1.2 intra-martのメニュー構成	4
1.2.1 メニューの表示	4
1.2.2 intra-martのホーム画面	5
1.3 サンプルプログラムの実行	6
1.3.1 ソースリストと実行画面を見るには	6
1.3.2 本書のサンプルを実際に作成してみるには	6
1.4 APIリストの見方	7
1.4.1 APIリストの格納場所	7
1.4.2 APIリストの操作	7
1.5 2つのWebアプリケーションモデル	8
1.5.1 スクリプト開発モデル	8
1.5.2 JavaEE開発モデル	8
1.5.3 2つの開発モデルで構築したアプリケーションを混在可能	8
1.5.4 各ファイルの保存場所	9
1.5.4.1 intra-mart WebPlatform (Resin) の場合	9
1.5.4.2 intra-mart WebPlatform (JBoss) およびintra-mart AppFrameworkの場合	9
1.6 intra-martのアプリケーション開発概要	10
1.6.1 プrezentation・ページ	10
1.6.2 ファンクション・コンテナ	11
1.7 スクリプト開発モデルでの開発	12
第2章 スクリプト開発モデルのプログラミングの基礎	13
2.1 Hello Worldを作ろう	14
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	14
2.1.2 ファンクション・コンテナ (.js) の作成	15
2.1.3 アプリケーション・プログラムの実行	16
2.2 ページ間にまたがるデータの共有	17
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	17
2.2.1.1 送信側のHTMLファイルの準備 (input.html)	17
2.2.1.2 表示側のHTMLファイルの準備 (hello.html)	18
2.2.2 ファンクション・コンテナ (.js) の作成	19
2.2.2.1 受取側のJavaScriptファイルの作成 (input.js)	19
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)	19
2.2.3 アプリケーション・プログラムの実行	19
2.3 データベースからデータを取得する	22
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	23
2.3.2 ファンクション・コンテナ (.js) の作成	24
2.3.3 アプリケーションの実行	25
2.4 取得したデータの一覧表示	26
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	26
2.4.2 ファンクション・コンテナ (.js) の作成	26
2.4.3 アプリケーションの実行	27
2.4.4 項目の拡張	27
2.5 データの登録・更新・削除	28
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	28

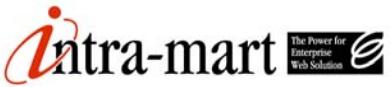
2.5.2 ファンクション・コンテナ (.js) の作成	29
2.5.3 アプリケーションの実行	30
<b>第3章 さまざまなコンポーネント群(im-BizAPI)の利用</b>	<b>32</b>
<b>3.1 Storage Serviceの利用方法</b>	<b>33</b>
3.1.1 ファイル・アップロード	33
3.1.2 ファイルリストの表示	35
3.1.3 ファイル・ダウンロード	37
3.1.4 ファイルの削除	39
<b>3.2 メール送信</b>	<b>42</b>
3.2.1 メール送信フォームの作成	42
3.2.2 添付ファイル付きメールの送信	44
<b>3.3 拡張&lt;IMART&gt;タグ機能の使用例</b>	<b>47</b>
3.3.1 タグの定義と登録	47
3.3.2 拡張<IMART>タグの利用	48
<b>3.4 ユーザ定義関数の登録と利用</b>	<b>49</b>
3.4.1 グローバル関数としての登録・呼び出し	49
<b>3.5 JavaClassとの連携</b>	<b>51</b>
3.5.1 標準JavaClassとの連携方法	51
3.5.1.1 標準JavaClass連携時の問題点	53
3.5.1.2 自作JavaClassとの連携方法	53
3.5.1.3 intra-mart起動時の設定方法	53
3.5.1.4 自作JavaClass側の記述方法	53
3.5.1.5 サーバサイドJavaScript側の記述方法	54
3.5.6 EJBとの連携	56
3.6.1 EJBコンポーネントの作成	56
3.6.2 JavaScriptからの呼び出し	56
<b>3.7 外部プロセスの呼び出し</b>	<b>57</b>
<b>3.8 XML形式のデータを扱う</b>	<b>58</b>
3.8.1 XMLパーサーとデータの取得	58
3.8.2 XML形式データの受信方法	58
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	59
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	59
3.8.3 XML形式データの送信方法	60
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	61
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	62
<b>3.9 E4Xの利用方法</b>	<b>63</b>
3.9.1 E4Xとは？	63
3.9.2 XMLオブジェクトの作成	63
3.9.2.1 XML構文からXMLオブジェクトを作成する	63
3.9.2.2 文字列からXMLオブジェクトを作成する	63
3.9.3 値の取得	63
3.9.4 子ノードの取得	64
3.9.5 ノードの追加	64
3.9.6 ノードの削除	64
3.9.7 変数の挿入	65
<b>3.10 JSSP-RPCについて</b>	<b>66</b>
3.10.1 動作イメージ	66
3.10.2 JSSP-RPC 通信エラーオブジェクトについて	66
3.10.3 JSSP-RPC サンプルプログラム（同期通信）	67
3.10.3.1 クライアントサイドのHTMLソース	67

3.10.3.2 サーバサイドのJSソース 「jssp_rpc_test/sample1.js」	67
3.10.4 JSSP-RPC サンプルプログラム（非同期通信）	68
3.10.4.1 クライアントサイドのHTMLソース	68
3.10.4.2 サーバサイドのJSソース 「jssp_rpc_test/sample2.js」	69
3.11 デバッグ手順	70
3.11.1 デバッグ例	70
3.11.2 デバッグAPIの利用方法	71
3.12 単体テスト環境（im-JsUnit）	73
3.12.1 im-JsUnit概要	74
3.12.2 テストケースの実行順序	75
3.12.3 テストケースの作成	76
3.12.3.1 評価関数の使用方法	77
3.12.3.2 テストケース作成における注意点	77
3.12.4 テストランチャーの作成	78
3.12.5 単体テストの実行	78
3.13 JavaScriptコンパイラ機能について	79
3.13.1 実行時のファンクション・コンテナ検索手順	81
3.13.2 仕様詳細	82
3.13.3 コンパイラとは直接関係ない部分の仕様	83
3.13.4 制約	83
3.13.4.1 ファイルサイズによる制約	83
3.13.4.2 プログラムの書き方による制約	83
3.14 im-JavaEE Frameworkとの連携	85
3.15 サンプル・アプリケーション	86
3.15.1 データベースへのサンプルデータの登録	86
3.15.2 「勤怠管理」アプリケーションの操作	86
3.15.2.1 勤怠登録	86
3.15.2.2 勤怠修正	88
3.15.3 ワークフロー・モジュールとの連携	89
3.15.3.1 申請画面	89
3.15.3.2 フロー情報画面	89
3.15.3.3 承認画面	90
3.16 モジュールの組み込みと操作	91
3.16.1 ユーザインターフェース層	92
3.16.1.1 画面共通モジュール	92
3.16.1.2 グラフ描画モジュール（プレゼンテーション・ページ）	93
3.16.1.3 アクセスコントローラモジュール	95
3.16.2 ビジネスロジック層	96
3.16.2.1 アプリケーション共通モジュール	96
3.16.2.2 メール連携モジュール（ファンクション・コンテナ）	96
3.16.2.3 外部ソフトウェア接続モジュール	98
3.16.2.4 ERP連携モジュール	99
3.16.3 業務基盤ツール	99
3.16.3.1 アクセスセキュリティ・モジュール	99
3.16.3.2 ワークフロー・モジュール	100
3.16.3.3 ビジネスプロセスワークフローモジュール	100
3.16.3.4 バッチ管理モジュール	101
3.16.3.5 ポータルモジュール	101
3.16.3.6 ViewCreator	102
3.17 ユニットの組み込みと操作	104
3.17.1 アプリケーション共通マスタunit	104

3.17.2 カレンダーunit	104
3.17.2.1呼び出し方法	105
3.17.2.2 カレンダーデータの受け取り方法	105
3.17.2.3 カレンダー拡張タグとカレンダーモジュール	106
3.17.3 ファイルダウンロードunit	106
3.17.3.1 ダウンロードの方法	106
3.17.3.2 ファイル拡張子とMIMEタイプ	106
3.17.4 ファイルアップロードunit	106
3.17.4.1 プレゼンテーション・ページとの連携	107
3.17.4.2 情報の取得方法	107
3.17.5 ツリー表示unit	108
3.17.6 i-mode unit	108
3.17.6.1 ページ管理マスタメンテナンスでのi-modeの設定	108
3.17.6.2 i-modeのアドレスとパスワードの設定	109
3.17.6.3 i-mode用外出設定	110
3.18 エクステンション・モジュールの組み込みと操作	111
3.18.1 帳票印刷モジュール拡張	111
3.18.1.1 IM-PDFデザイナー	111
3.18.1.2 IM-X Server	113
3.18.2 アクセスセキュリティ・モジュール拡張	114
3.18.2.1 IM-SecureSignOn(セキュア・サイン・オン)	114
3.18.2.2 IM-SecureBlocker	116
3.18.3 ワークフロー・モジュール拡張	117
3.18.3.1 IM-ワークフローデザイナー	117
3.18.3.2 IM-FormatCreator	117
3.18.3.3 IM-EX申請システム	117
3.18.3.4 IM-SonicESB	118
3.18.4 外部ソフトウェア連携ソリューション	119
3.18.4.1 統合検索ソリューション	119
3.18.4.2 マルチデバイスソリューション	119
3.19 その他の機能	120
3.19.1 ライブラリ	120
3.19.2 検索ストリーミング機能	120
3.19.3 ソースセキュリティ機能	120
3.19.4 アプリケーション・ロック機能	120
3.19.5 一意情報の取得機能	121
3.19.6 データベースのストアドプロシージャの呼び出し	121
3.19.7 ファイル操作	121
3.19.8 データベース操作	122
3.19.9 LDAPとの連携	122
3.19.10 国際化対応	122
3.19.11 標準画面の作り方(共通画面デザイン)	123
3.19.12 ポータル画面の利用	123
3.19.13 ショートカットアクセス機能	123
第4章 Appendix	126
4.1 メッセージ設定	127
4.2 予約語一覧	128
4.3 制限事項	129
4.3.1 ファイル名称	129
4.3.2 ID、コード	129







The Power for  
Enterprise  
Web Solution

**intra-mart WebPlatform/  
AppFramework**

# 第1章 イントロダクション

# 1.1

# intra-martの起動と終了

Webベースの開発ツールであるintra-martは、通常のコンピュータソフトとは起動と終了の方法が異なります。Webベースのアプリケーションでは、ブラウザ上から本システムへ「ログイン」で使用できるようになり、終了時は「ログアウト」で終了させます。



## 1.1.1 intra-martへのログイン

intra-martへのログインは、システム管理者、ログイングループ管理者、一般ユーザによってURLが異なります。以下、一般ユーザのログインの手順を説明します。なお、システム管理者、ログイングループ管理者のログインに関しては、「アドミニストレータガイド」を参照してください。

1

ブラウザを起動し、intra-martのURLを入力します。

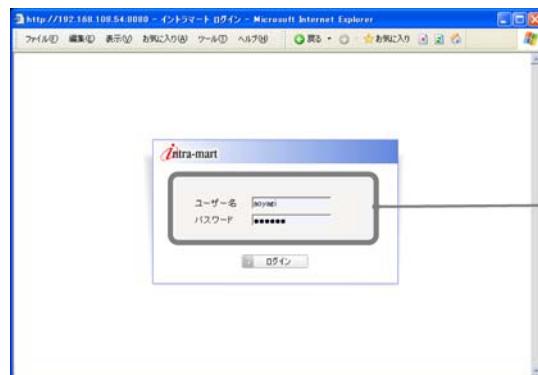
画面上には、intra-martのログイン画面が表示されます。

```
intra-mart WebPlatform
  (スタンドアロン)   :http://マシンアドレス/imart/(ログingroup名).portal
  (分散システム)   :Webサーバコネクタの登録内容に合わせたURL
intra-mart AppFramework
  :Webアプリケーションサーバに対するintra-martの登録内容に合わせたURL
```

- intra-mart WebPlatformをスタンドアロン形態で運用する場合、アクセスするURLのポート番号(Webサーバとしてのポート)は、インストール時に指定することができます。
- URLは、ブラウザのブックマークに登録しておくと便利です。
- この他に、「ログイン画面なしで自動認証する方法」(→P. 3 Column参照)が用意されています。

2

ログイン画面で、ユーザコードと、パスワードを入力し、[ログイン]ボタンをクリックします。



<intra-martのログイン画面>



- ログイン画面及びログイン初期画面のソースは、以下の場所に用意されています。  
%ResourceService%/pages/platform/src/system/securityフォルダ



&lt;intra-martの初期画面&gt;



- 初期画面のメインページ(右フレーム内)は、ポータル画面となっています。新しくポートレットを作成してポータル機能に登録することにより、様々な情報を初期表示することができます。



## 1.1.2 intra-martからのログアウト

ログイン画面に戻るとログアウトしたことになり、intra-martを終了できます。ログイン画面に戻るには、画面左のメニュー上部に用意されている [LOGOUT] ボタンをクリックします。



- メニューの[LOGOUT]ボタンをクリックせずにブラウザを終了した場合や、他のページに移動してintra-martの画面から離れてしまった場合、intra-martサーバ内ではセッションがタイムアウトするまでログイン状態を継続しているものとみなされます。必ずメニューの[LOGOUT]ボタンをクリックしてください。



Column

## ログイン画面なしで自動認証する方法

intra-martにログインする際のURLに、次のようにユーザコードとパスワードを含めると、ログイン画面なしで自動認証されます。

### ■システム管理者

`http://intramart/imart/system.admin?im_user=ユーザコード&im_password=パスワード`

### ■ログingroup管理

`http://intramart/imart/ログingroup名.manager?im_user=ユーザコード&im_password=パスワード`

### ■一般ユーザ

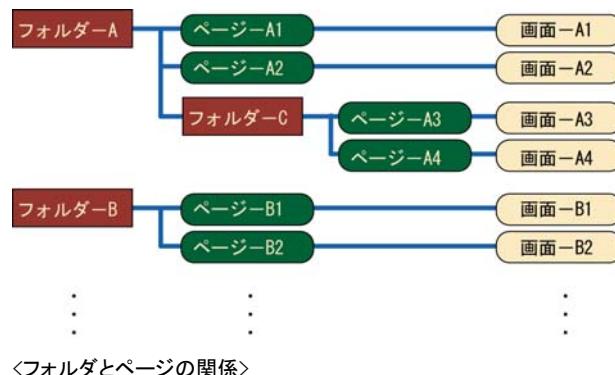
`http://intramart/imart/ログingroup名.portal?im_user=ユーザコード&im_password=パスワード`

# 1.2

# intra-martのメニュー構成

intra-martのメニュー表示形式は、「フォルダ」という大分類と「ページ」から構成されていて、ページを選択すると該当する画面が表示されます。フォルダはどの階層にでも登録することができ、その配下のページをまとめるために利用します。

フォルダおよびページは、ログイン（起動）したユーザのアクセス権に応じたものだけが表示されるので、本マニュアル上の画面と実際の画面が異なることがあります。



## 1.2.1 メニューの表示

intra-martのメニュー表示では、画面左フレームにツリー構造のメニューが表示されます。大項目である「フォルダ」をクリックすると、そこに属する「ページ」がツリーに表示されます。ここで、「ページ」を選択すると、これに該当する画面内容が表示されます。



<メニュー表示画面の例>



## 1.2.2 intra-martのホーム画面

ホーム画面は、intra-martログイン時に表示される初期画面です。メニューを選択して、各種画面が表示されている場合には、画面右の[HOME]ボタンをクリックするとホーム画面が表示されます。この画面からintra-martのポータル画面への切り替えが行えます。画面の切り替えは、右画面の左上のタブで切り替えることができます。



[HOME]ボタン

ホーム画面に戻ります。ホーム画面はintra-martログイン時に表示される画面で、ポータル画面が設定されているときには、この画面で切り替えることができます。

[LOGOUT]ボタン

intra-martを終了して、ログイン画面に戻ります。

[MENU ON/OFF]ボタン

左側のメニューをOn/Offします。メニューをOffにすると、画面が広くなり作業が容易になります。なお、[属性設定]メニューで、ログイン直後のメニューのOn/Off状態を変更することもできます。

[ポータル切り替え]タブ

設定されているポータル画面を切り替えることができます。ポータル画面の詳細は、アドミニストレータガイド第2章「10 ポータルの設定と操作」と、同じく 第3章「4 ポータルの利用」を参照してください。

マイメニュー

よく利用するメニューを「マイメニュー」として登録できます。メニューは、画面上部または左メニュー最上部の2箇所を切り替えることができます。

# 1.3

# サンプルプログラムの実行

本書の以降の章では、intra-mart WebPlatform/AppFrameworkに付属しているサンプルプログラムに対応して、プログラミングの基礎を体験していただけるように構成されています。本書を読みながらソースプログラムを表示し、実行してみることができます。また、ユーザが一からサンプルを作成して実行できる環境も用意されています。

## スクリプト開発モデル

サンプルプログラム： [サンプル]-[スクリプト開発モデル]-[チュートリアル]-[初級]

ソースファイル : %Storage Service%/sample/tutorial/beginner/source/



## 1.3.1 ソースリストと実行画面を見るには

[初級] というページの「ソース」というリンクを選択すると、各節で説明しているソースリストを画面に表示することができます。また、「実行画面」というリンクを選択すると、実際に実行した画面を見ることができます。



## 1.3.2 本書のサンプルを実際に作成してみるには

intra-martをインストールすると、「sample」というフォルダができています。この中に、「スクリプト開発モデル」および「JavaEE開発モデル」用のフォルダが用意されており、以降の章のすべてのサンプルと同じ名前のテンプレートファイルが入っています。このファイルを利用してサンプルを作成すると、[初級] ページ下部のリンクを選択するだけで実際に実行することができるようになっています。また、「tutorial/beginner/example」というフォルダには、完成したサンプルが入っていますので、「tutorial/beginner/training」フォルダに全ファイルをコピーして実行することもできます。



<チュートリアル[初級編]の画面例>



- ここで紹介したサンプルのほかにも、メニューの[サンプル]フォルダにはプログラミングの参考になる勤怠管理などのサンプルプログラムが用意されています。こちらも合わせてご利用ください。

# 1.4 APIリストの見方

APIリストには、intra-mart WebPlatform/AppFrameworkで提供される各機能やAPIの仕様について記載されています。開発の際には、参考資料としてご活用ください。APIリストは、HTML形式およびWindowsの標準Help形式が用意されています。Windowsの標準Help形式では、【検索】タブで全文検索が行えます。



## 1.4.1 APIリストの格納場所

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_afw/development/apilist_v61.zip
Windowsの標準Help形式	: iwp_afw/development/apilist_v61.chm

Windowsの標準Help形式のAPIリストを表示させると、次のようなメニューが表示されます。

The screenshot shows the Windows standard Help interface for the 'Coding Rules' topic under the 'J2EE-based Development Model' section. The left pane displays a hierarchical table of contents with categories like 'Document', 'Developer's Guide', and 'Trouble Shooting'. The right pane contains the main content area with the title 'Coding Rules (J2EE-based Development Model)' and a 'Note' section. The note text is as follows:

このページで記載されている内容は、intra-mart アプリケーションベースモジュールの各画面で同じくカスタマイズする際の必要事項になりますので、十分に理解した上で、intra-mart アプリケーションの構築を行なってください。

J2EEベース開発モデルに關しては、JAVA、JSP および Servlet 等、関連技術のレポートは後で詳しく説明します。

これ以外に intra-mart J2EEベース開発モデルでのルールは、以下の通りとなります。

**ルール**

1. intra-mart ベースモジュールで提供している機能において、すべてのIDに同じで、以下に示す文字を含む名称を与えることはできません。  
\$/.;,\*?"<>|&#+[]{}{}(space)(tab)(全角文字)
2. ファンクション・コントナ内で [install\_directory]/conf/imart.xml を読み込んだり書き込んだりしてはいけません。

設定ファイルを簡単に変更することは、思わぬ誤動作の原因となります。

なお、[install\_directory]/conf/imart.xml ファイル内の設定内容を変更した場合には、intra-mart システム全体の再起動が必要です。ただし、intra-mart Administrator(サーバ管理ツール)上で設定内容を変更した場合は、intra-mart Administrator(サーバ管理ツール)から該当のサーバを再起動するだけで新しい設定内容が反映されます。

<APIリスト>



## 1.4.2 APIリストの操作

左側の【目次】タブから、各カテゴリーのモジュール名をクリックすることで、各モジュール内容が表示されます。【検索】タブを利用すると、目的語で検索することができます。【キーワード】タブに関しては、現在使用していません。

intra-martには「スクリプト開発モデル」と「JavaEE開発モデル」の2つの開発モデルが用意されています。これらの開発モデルは、それぞれ特徴を持ち、用途に応じて使い分けることができます。また、2つのモデルで開発したWebアプリケーションを同一のシステム内に混在させて実行することもできます。どちらの開発モデルもアクセスセキュリティモジュール、ワークフローモジュールなど、intra-martが標準で用意しているフレームワークを利用することができますので、高い生産性を誇ります。



### 1.5.1 スクリプト開発モデル

ホームページの作成に利用しているHTMLと(サーバサイド)JavaScriptを利用してWebシステムを構築する開発モデルです。そのため、初心者でも始められる簡易性をもち、要員のシステム教育コストを大きく削減することができます。これまで簡単なホームページしか作成したことがなかった方でも約2週間～1ヶ月ほどでマスターし、データベースと連動したWeb業務画面を作成することができます。また、ホームページの作成更新と同様に簡易に開発ができますので、頻繁に更新するような複雑なWebシステムにも柔軟に対応することができます。

さらにJavaScriptからは、開発者のテクニカルスキルにあわせて、Java(Class、EJB)、C++、ストアドプロシージャなどが簡単に呼び出すことができます。



- スクリプト開発モデルに関しては、本書で解説しています。



### 1.5.2 JavaEE開発モデル

JavaEE(Java EnterpriseEdition)とは、米サン・マイクロシステムズの提唱によるエンタープライズシステム向けのJavaプラットフォームです。Servlet、JSP (Java Server Pages)、EJB (Enterprise Java Beans)などから構成され、MVCモデル(Model-View-Controller)でシステム構築をおこなっていきます。特に高トランザクションが集中する処理に適しています。



- JavaEE開発モデルに関しては、「プログラミング・ガイド JavaEE開発モデル」を参照してください。



### 1.5.3 2つの開発モデルで構築したアプリケーションを混在可能

intra-mart WebPlatform/AppFrameworkでは、これら2つの開発モデルで構築したアプリケーションを同一システム内で混在させることもできます。そのため、開発予算が限られ短期間開発の必要なWebシステムはスクリプト開発モデルを主体に開発を進め、その中でトランザクションの集中する部分のみを切り出してJavaEE開発モデルで開発するといった現実的かつ柔軟な開発スタイルをとることができます。



- プログラムを作成するにあたっては、プログラミングガイド(スクリプト開発モデル編およびJavaEE開発モデル編)をお読みいただくとともに、アドミニストレータガイドもお読みください。その他、「im-JavaEE Framework仕様書」もドキュメントCDに用意されています。あわせてご活用ください。

ドキュメントCDの [iwp\\_afw/specification/im-javaee\\_framework-spec\\_v61.pdf](#)



## 1.5.4 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。



### 1.5.4.1 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/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/)

#### ■ JavaEE開発モデルのプログラム(JSP)(JSP ファイル(.jsp, .xtp))

%Application Runtime%(スタンドアロンの場合はサーバモジュール)/doc/ディレクトリ以下

#### ■ JavaEE開発モデルのプログラム(Servlet)(JAVAクラスファイル(.class))

%Application Runtime%(スタンドアロンの場合はサーバモジュール)/doc/ディレクトリ以下の該当ディレクトリ内WEB-INF/classes/以下 (または、クラスパスに設定されているディレクトリ内)

#### ■ 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



### 1.5.4.2 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/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/)

#### ■ JavaEE開発モデルのプログラム(JSP)(JSP ファイル(.jsp))

%Application Runtime%(スタンドアロンの場合はサーバモジュール)/ doc/ディレクトリ以下

#### ■ JavaEE開発モデルのプログラム(Servlet)(JAVAクラスファイル(.class))

%Application Runtime%(スタンドアロンの場合はサーバモジュール)/doc/ディレクトリ以下の該当ディレクトリ内 WEB-INF/classes/以下 (または、クラスパスに設定されているディレクトリ内)

#### ■ Storage Serviceにより一元管理されるファイル群

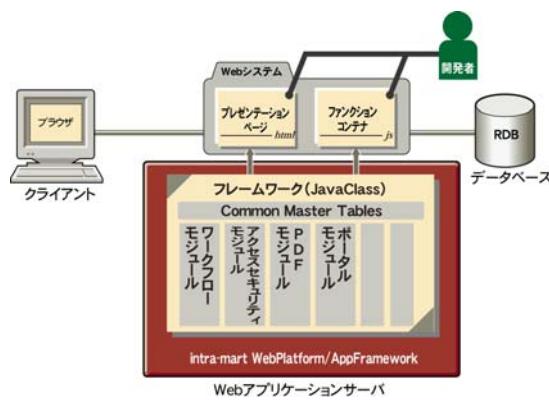
%Storage Service%/storage/ディレクトリ内

# 1.6

## intra-martのアプリケーション開発概要 (スクリプト開発モデルの場合)

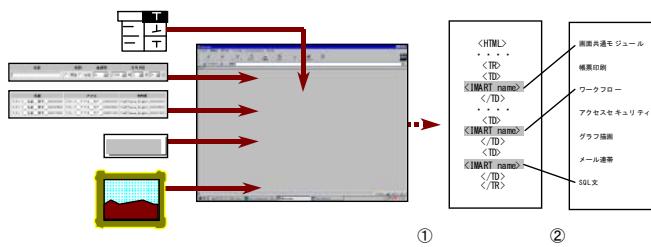
intra-mart WebPlatform/AppFrameworkを使ったアプリケーションの開発において、開発者はブラウザ上に表示するコンテンツと、サーバ上で動作するビジネスロジックを作成することになります。

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。



### 1.6.1 プrezentashon・ページ

プレゼンテーション・ページは、ユーザインター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アリッヂ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、ユーザインター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。



- ① 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ー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。
- ② 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くます。

```
< 社員マスタからのデータ取得用 HTML (一覧表示用) 一項目を拡張 >
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>
```

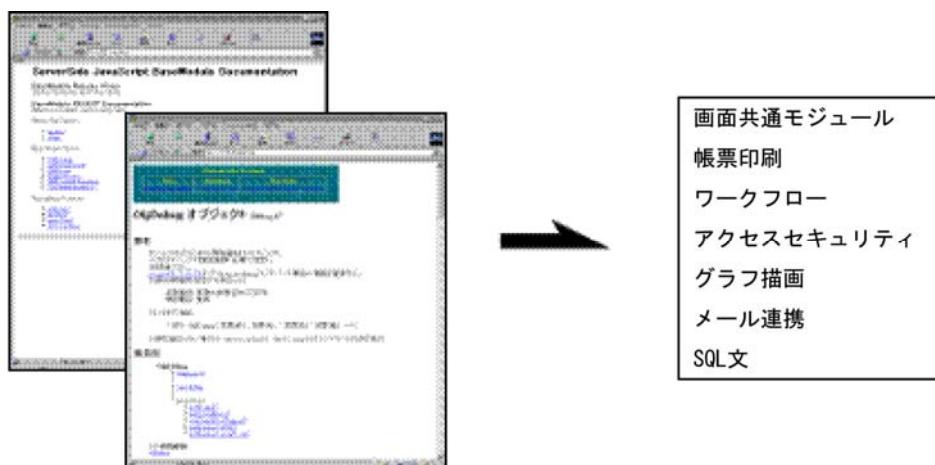
intra-martの独自拡張タグ<IMART>を利用して、各種モジュールを呼び出していくます。



## 1.6.2 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。



本書第2章以降には、次のような内容を記述しています。スクリプト開発モデルで開発するにあたり、ご利用ください。

### ■ 第2章 スクリプト開発モデルのプログラミングの基礎

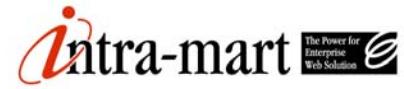
- 1 Hello Worldを作ろう
- 2 ページ間にまたがるデータの共有（セッション管理）
- 3 データベースからデータを取得する
- 4 取得したデータの一覧表示
- 5 データの登録・更新・削除

### ■ 第3章 さまざまなコンポーネント群(im-BizAPI)の利用

- 1 Storage Serviceの利用方法
- 2 メール送信
- 3 拡張<IMART>タグ機能の使用例
- 4 ユーザ定義関数の登録と利用
- 5 JavaClassとの連携
- 6 EJBとの連携
- 7 外部プロセスの呼び出し
- 8 XML形式のデータを扱う
- 9 E4Xの利用方法
- 10 JSSP-RPCについて
- 11 デバッグ手順
- 12 単体テスト環境(im-JsUnit)
- 13 JavaScriptコンパイラ機能について
- 14 im-JavaEE Frameworkとの連携
- 15 サンプル・アプリケーション
- 16 モジュールの組み込みと操作
- 17 ユニットの組み込みと操作
- 18 エクステンション・モジュールの組み込みと操作
- 19 その他の機能

### ■ Appendix

- 1 メッセージ設定
- 2 予約語一覧
- 3 制限事項



intra-mart WebPlatform/  
AppFramework

## 第2章 スクリプト開発モデルのプログラミングの基礎

# 2.1

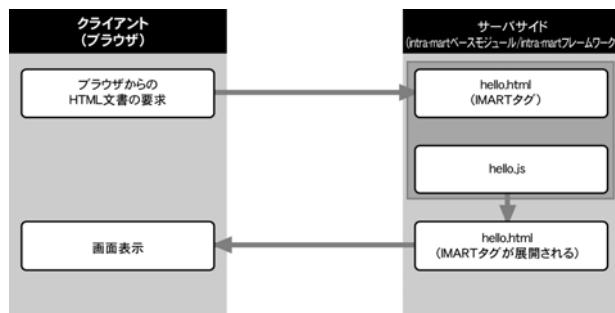
# Hello Worldを作ろう

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。



- 本章で紹介するサンプル事例は、すべてintra-martの[サンプル]-[スクリプト開発モデル]-[チュートリアル]メニュー内に登録されています。本書の以下の節は、それぞれ次のチュートリアルに対応した解説になっています。
  - 第2章「1 Hello Worldを作ろう」～「5 データの登録・更新・削除」 → 初級ページ
  - 第3章「1 Storage Serviceの利用方法」 → ファイル操作ページ
  - 第3章「2 メール送信」 → メール送信ページ

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。

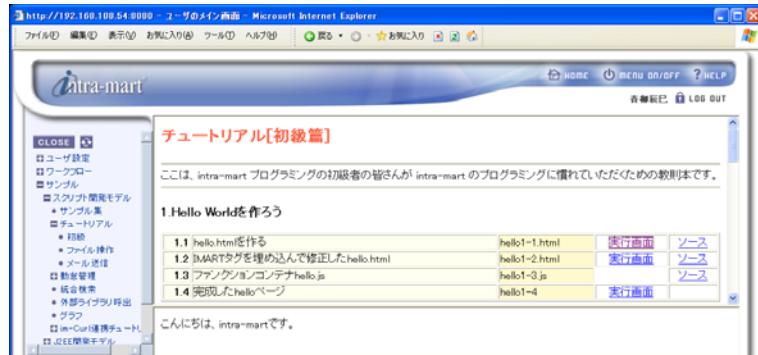


## 2.1.1 ベースとなるプレゼンテーション・ページ(.html)の作成

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、imartフォルダ下のSampleフォルダに保存します。

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



<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>
1:  <HTML>
2:  <BODY>
3:  こんにちは、<IMART type="string" value=nameValue></IMART>です。
4:  </BODY>
5:  </HTML>
```



Column

<IMART type="string">

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。



## 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) >
1:  // HTMLへ渡す値の宣言
2:  var nameValue;
3:
4:  // init関数の定義
5:  function init()
6:  {
7:      nameValue = "イントラマート";           // HTMLへ渡す値を設定します
8:  }
```



- 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。

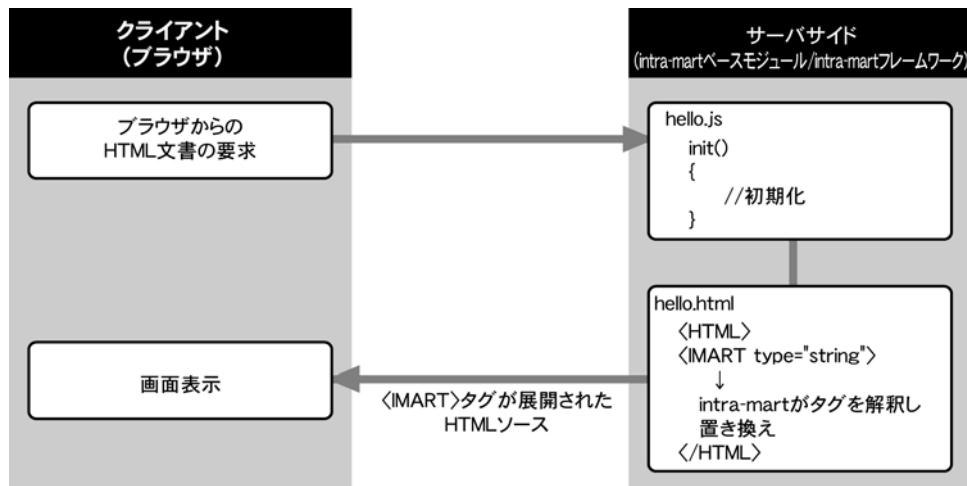


### 2.1.3 アプリケーション・プログラムの実行

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。



<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、ログインループ管理者でログインし、[ログインループ管理]-[メニュー管理]-[メニュー設定]でページの登録をする必要があります。アプリケーションの登録は、アドミニストレータガイドの第2章「7 アプリケーションの登録」を参照してください。

## 2.2

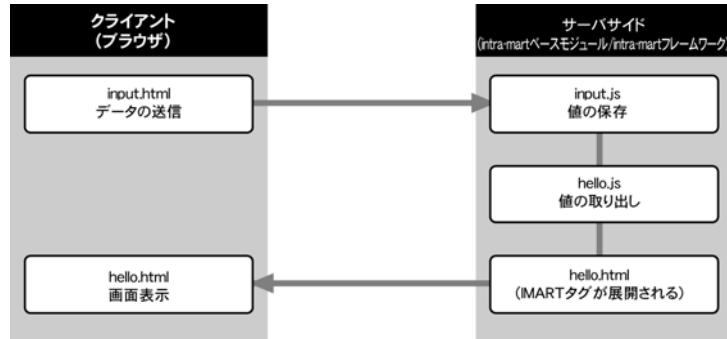
# ページ間にまたがるデータの共有

(セッション管理)

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成します。

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。ファイル名はinput.htmlです。

```
<送信フォームinput.html>
1: <HTML>
2: <BODY>
3: 名前を入力してください。<BR>
4: <FORM method="POST">
5:   <INPUT type="text" name="yourname"><BR>
6:   <INPUT type="submit" value="送信">
7: </FORM>
8: </BODY>
9: </HTML>
```



<input.html実行時の画面>

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、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>
1:  <HTML>
2:  <BODY>
3:  名前を入力してください。<BR>
4:  <IMART type="form" method="POST" action="inputName" page="sample/user1/hello">
5:    <IMART type="input" style="text" name="yourname"></IMART><BR>
6:    <IMART type="submit" value="送信"></IMART>
7:  </IMART>
8:  </BODY>
9:  </HTML>
```



### 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>
```



Column

<IMART type="link">

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。

## 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JavaScript>
1: // inputName関数の定義
2: function inputName(request)
3: {
4:     // 受け取った値をClientオブジェクトに保存する
5:     Client.set("nameValue", request.yourname );
6: }
```



### 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7行目を次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>
7: nameValue = "イントラマート";           // HTMLへ渡す値を設定
    ↓
7: 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

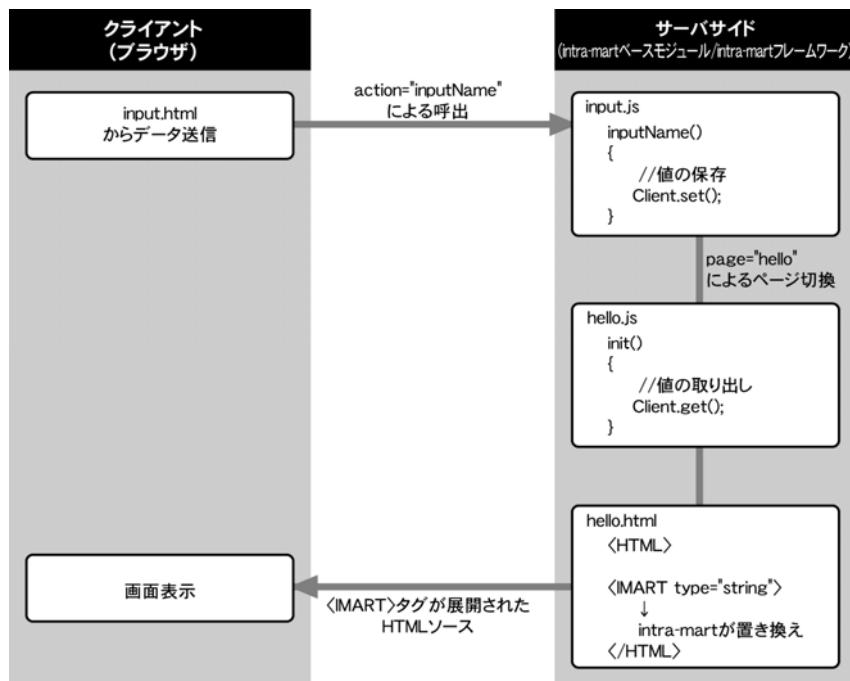
作成したプレゼンテーション・ページとファンクション・コンテナからなるアプリケーションを実行すると、送信側input.htmlで入力して送信したデータ「ユーザ1」が、受信側hello.htmlに表示されます。



<送信側input.html>



<受信側hello.html>



## Column

### <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。



## Column

### 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。



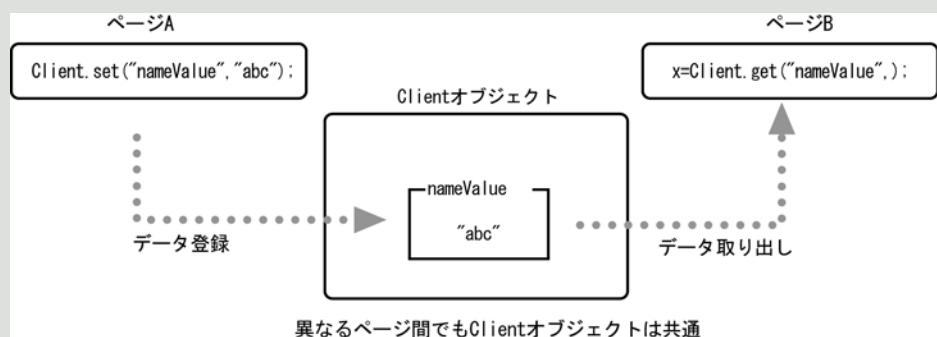
Column

## 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。



Column

## 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/h t t p.xml（基本設定ファイル）で変更することができます。conf/h t t p.xmlの編集に関しては、設定ガイドを参照してください。

## 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。

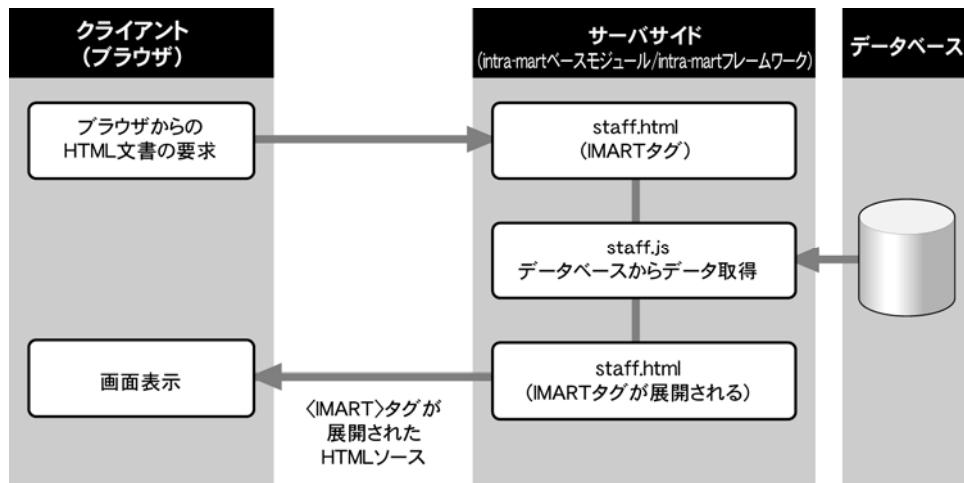
前章の手順に従って、サーバ側で動的に生成されるページを作成していきます。

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>



- ここでサンプルとして利用する社員マスタ(m\_sample\_stf)は、intra-mart WebPlatformをインストールした後、システム管理者画面の[ライセンス]で作成できます。



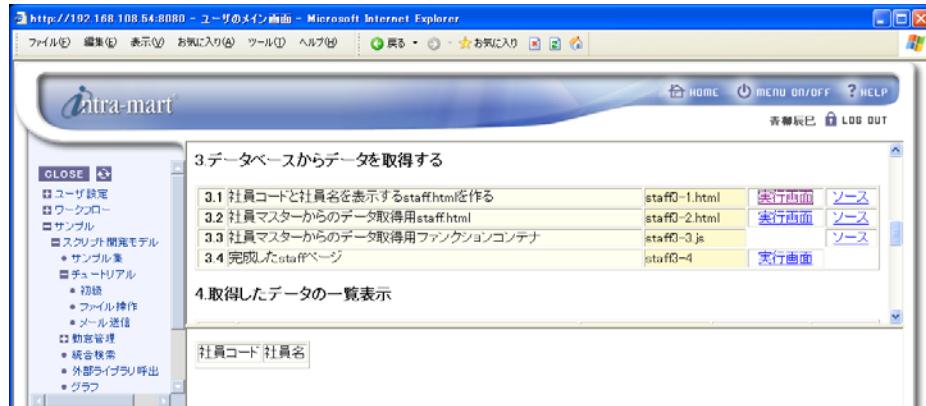


### 2.3.1 ベースとなるプレゼンテーション・ページ(.html)の作成

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できるので、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>
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: </TR>
8: <TR>
9:   <TD></TD>
10:  <TD></TD>
11: </TR>
12: </TABLE>
13: </BODY>
14: </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。

```
<社員マスターからのデータ取得用staff.html>
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: </TR>
8: <TR>
9:   <TD><IMART type="string" value=staff_code></IMART></TD>
10:  <TD><IMART type="string" value=staff_name></IMART></TD>
11: </TR>
12: </TABLE>
13: </BODY>
14: </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ファイル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>
1: // HTMLへ渡す値を宣言します
2: var staff_code;
3: var staff_name;
4:
5: // init関数の定義
6: function init(request)
7: {
8:     var objData = false; // データベースから取得したデータ格納用
9:
10:    // データベースから社員データを全て取得します
11:    objData = DatabaseManager.select("SELECT * FROM m_sample_stf");
12:
13:    // 1件目のレコードからHTMLへ渡す値を設定します
14:    staff_code = objData.data[0].staff_cd;
15:    staff_name = objData.data[0].stf_name_kanji;
16:
17: }
```



- staff\_cd、stf\_name\_kanjiはデータベースの列名です。



Column

### DatabaseManagerオブジェクトー1

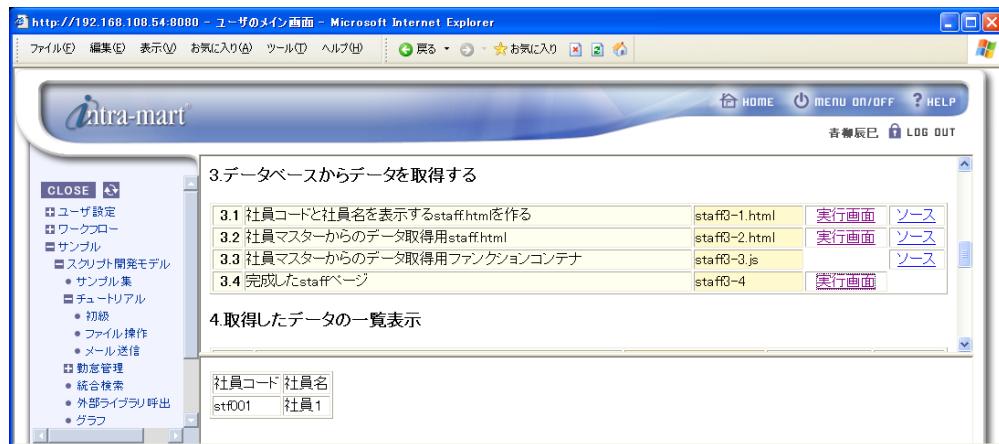
データベースへのアクセスはDatabaseManagerオブジェクトを通して簡単に行えます。  
この例では、DatabaseManagerオブジェクトの「select」メソッドを呼び出しています。このようにパラメータにSQL形式のSELECT文を記述してデータベースからデータを取得することができます。  
取得したデータはオブジェクトとして返されます。データベースから取得したレコードは、selectメソッドが返すオブジェクトのもつ配列プロパティとして保存されています。名称は「data」です。selectメソッドが返すオブジェクトには他にも、取得できたレコード数を保有する「countRow」などがあります。

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。

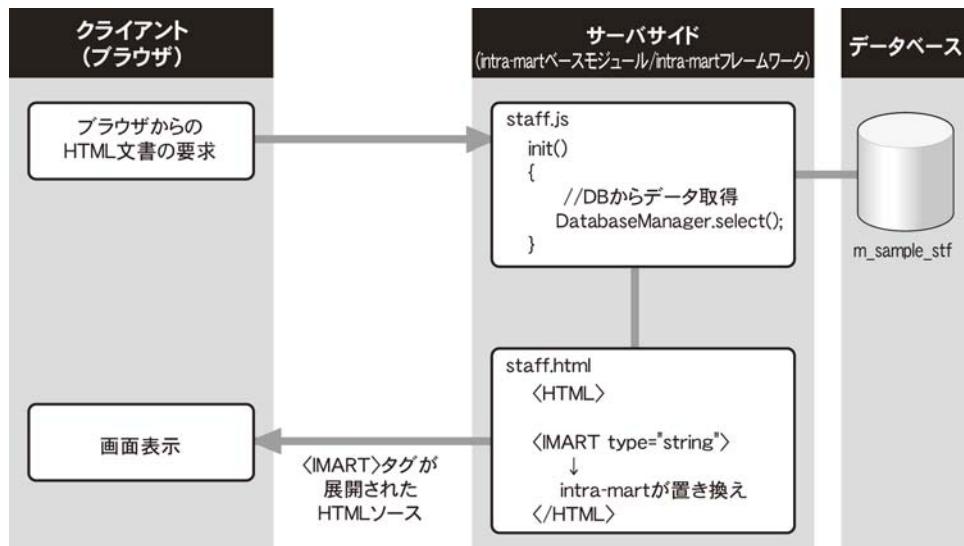


### 2.3.3 アプリケーションの実行

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]、date[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）>
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: </TR>
8: <IMART type="repeat" list=staffList item="record">
9: <TR>
10:  <TD><IMART type="string" value=record.staff_cd></IMART></TD>
11:  <TD><IMART type="string" value=record.stf_name_kanji></IMART></TD>
12: </TR>
13: </IMART>
14: </TABLE>
15: </BODY>
16: </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）>
1: //HTMLへ渡す値を宣言します
2: var staffList;
3:
4: // init関数の定義
5: function init()
6: {
7:   var objData = false; // データベースから取得したデータ格納用
8:
9:   // データベースから社員データを全て取得します
10:  objData = DatabaseManager.select( "SELECT * FROM m_sample_stf" );
11:
12:  // HTMLへ渡すデータのリストを設定します
13:  staffList = objData.data;
14:
15: }
```

m\_sample\_stf は、社員  
マスターです。



### 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。

〈実行結果〉



### 2.4.4 項目の拡張

完成したアプリケーションに、「社員名（カナ）」の項目を追加してみましょう。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。

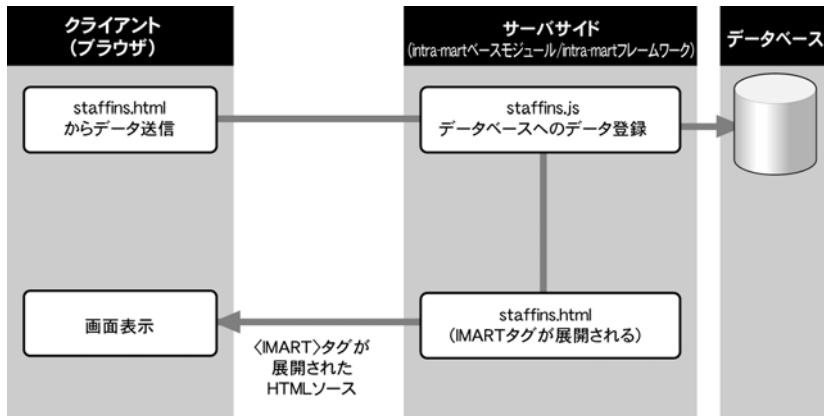
```
<社員マスターからのデータ取得用HTML（一覧表示用）一項目を拡張>
1:  <HTML>
2:  <BODY>
3:  <TABLE border="1">
4:  <TR>
5:    <TD>社員コード</TD>
6:    <TD>社員名</TD>
7:    <TD>社員名（カナ）</TD>                                ← 変更した行
8:  </TR>
9:  <IMART type="repeat" list=staffList item="record">
10: <TR>
11:   <TD><IMART type="string" value=record.staff_cd></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>      ← 変更した行
14: </TR>
15: </IMART>
16: </TABLE>
17: </BODY>
18: </HTML>
```

〈実行結果〉

## 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>
1: <HTML>
2: <BODY>
3: データを入力して登録ボタンを押してください。<BR>
4: <FORM method="POST">
5:   社員コード
6:   <INPUT type="text" name="staff_code"><BR>
7:   社員名
8:   <INPUT type="text" name="staff_name"><BR>
9:   <BR>
10:  <INPUT type="submit" value="登録">
11: </FORM>
12: <BR>
13: <IMART type="link" page="sample/user1/staff">戻る</IMART>
14: </BODY>
15: </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>
1: <HTML>
2: <BODY>
3: データを入力して登録ボタンを押してください。<BR>
4: <IMART type="form" method="POST" action="/insertStaffName">
5:   社員コード
6:   <IMART type="input" style="text" name="staff_code"></IMART><BR>
7:   社員名
8:   <IMART type="input" style="text" name="staff_name"></IMART><BR>
```

```

9:      <BR>
10:     <IMART type="submit" value="登録"></IMART>
11:   </IMART>
12:   <BR>
13:   <IMART type="link" page="sample/user1/staff">戻る</IMART>
14: </BODY>
15: </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>
1:  function insertStaffName(request)
2:  {
3:    var objRecord = new Object(); // レコード登録用オブジェクトを生成
4:    var result; // DBアクセスの結果
5:
6:    // レコードの作成
7:    objRecord.staff_cd      = request.staff_code;
8:    objRecord.stf_name_kanji = request.staff_name;
9:    objRecord.stf_name_kana  = request.staff_name_kana;
10:   objRecord.stf_name_eng   = request.staff_name_eng;
11:
12:   // DB処理の開始
13:   DatabaseManager.beginTransaction();
14:
15:   // レコードの挿入
16:   result = DatabaseManager.insert("m_sample_stf", objRecord);
17:
18:   // エラーチェック
19:   if (!result.error)
20:   {
21:     DatabaseManager.commit(); // 成功なのでコミットします
22:   }
23:   else
24:   {
25:     DatabaseManager.rollback(); // 失敗なのでロールバックします
26:   }
27: }

```



### 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。

5.1 staff.html に入力フォームページへのリンクを追加する	staff5-1.html	実行画面 ソース
5.2 データ更新用フォームstaffins.htmを作成する	staffins5-2.html	実行画面 ソース
5.3 ファンクションコンテナとの連携用staffins.html	staffins5-3.html	実行画面 ソース
5.4 データベースへの登録用ファンクションコンテナ	staffins5-4.js	ソース
5.5 実現したstaffページとstaffinsページ	staff5-5	実行画面 ソース
	staffins5-5	実行画面

データを入力して登録ボタンを押してください。

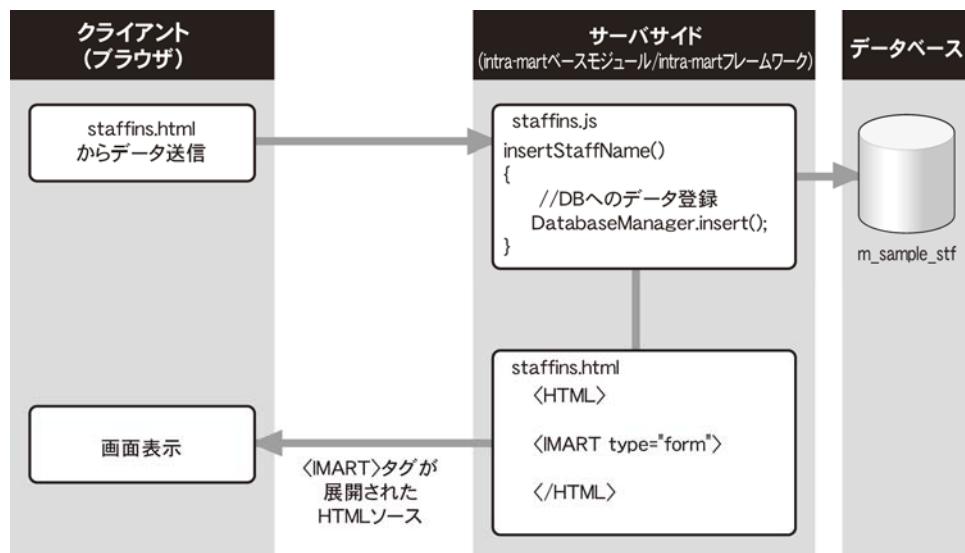
社員コード: 001  
社員名: ユーザー  
登録

<staffins.html実行画面>

5.1 staff.html に入力フォームページへのリンクを追加する	staff5-1.html	実行画面 ソース
5.2 データ更新用フォームstaffins.htmを作成する	staffins5-2.html	実行画面 ソース
5.3 ファンクションコンテナとの連携用staffins.html	staffins5-3.html	実行画面 ソース
5.4 データベースへの登録用ファンクションコンテナ	staffins5-4.js	ソース
5.5 実現したstaffページとstaffinsページ	staff5-5	実行画面 ソース
	staffins5-5	実行画面

社員コード: 社員名:  
001 ユーザー  
stf001 社員1  
stf002 社員2  
stf003 社員3  
stf004 社員4

<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。



Column

## DatabaseManagerオブジェクトー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
<code>update</code> メソッド	データの更新	<code>update</code> (表名, WHERE句の条件, 更新データ)
<code>remove</code> メソッド	データの削除	<code>remove</code> (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、

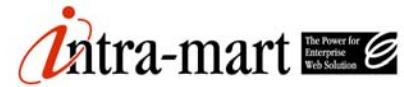
```
DatabaseManager.remove( "m_sample_stf", "staff_cd = 'stf0001' );
```

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。



intra-mart WebPlatform/  
AppFramework

## 第3章 さまざまなコンポーネント群(im-BizAPI)の利用

# 3.1

# 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成します。



## 3.1.1 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」>
1: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 //EN">
2:
3: <HTML>
4:   <HEAD>
5:     <TITLE>File Center</TITLE>
6:   </HEAD>
7:   <BODY bgcolor="WhiteSmoke">
8:     <H2>Upload</H2>
9:     <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
10:       <INPUT type="file" name="local_file">
11:       <INPUT type="submit" value=" send ">
12:     </IMART>
13:   </BODY>
14: </HTML>
15:
16: <!-- 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」>
1: // 画面初期化
2: function init(request) {
3:   var root = new VirtualFile("filebox");
4:
5:   // 保存ディレクトリの有無のチェック
6:   if(! root.isDirectory()){
7:     root.makeDirectories(); // ディレクトリ作成
8:   }
9:
10: }
11:
12: // 指定ファイルの受信
13: function action_upload(request) {
14:
15:   // パラメータ情報(=RequestParameter オブジェクト)を取得
16:   var parameter = request.getParameter("local_file");
17:
18:   // ファイルの中身を取得 (バイナリ)
19:   var fileData = parameter.getValueAsStream();
20:
21:   // ファイル名の取得
```

```

22:     var fileName = parameter.getFileName()
23:
24:     // ファイルの書き出し
25:     var vf = new VirtualFile("filebox/" + fileName);
26:     var res = vf.save(fileData);
27:
28: }

```

この画面では、以下のように動作します。

- 1: 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。
- 2: サーバでは、ファイルの内容の有無を確認します。
- 3: 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。
- 4: 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)>



Column

<INPUT type="file">

HTMLフォームの中で利用している<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。



Column

## 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。



### 3.1.2 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) >
1: <HTML>
2:   <HEAD>
3:     <TITLE>File Center</TITLE>
4:   </HEAD>
5:   <BODY bgcolor="WhiteSmoke">
6:     <H2>Upload</H2>
7:     <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
8:       <INPUT type="file" name="local_file">
9:       <INPUT type="submit" value=" send ">
10:      </IMART>
11:      <HR>
12:      <H2>File List</H2>
13:      <TABLE>
14:        <TR><TH>File Name</TH></TR>
15:        <IMART type="repeat" list=fList item="rec">
16:          <TR>
17:            <TD><IMART type="string" value=rec></IMART></TD>
18:          </TR>
19:        </IMART>
20:      </TABLE>
21:    </BODY>
22: </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。

```
<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>
1: var fList, rec;
2:
3: // 画面初期化
4: function init(request) {
5:   var root = new VirtualFile("filebox");
6:
7:   // 保存ディレクトリの有無のチェック
8:   if(! root.isDirectory()){
9:     root.makeDirectories();
10:   }
11:
12:   fList = root.files(); // ファイルリストの取得
13: }
14:
15: // 指定ファイルの受信
16: function action_upload(request) {
17:
18:   // パラメータ情報(=RequestParameter オブジェクト)を取得
19:   var parameter = request.getParameter("local_file");
20:
21:   // ファイルの中身を取得 (バイナリ)
22:   var fileData = parameter.getValueAsStream();
23:
24:   // ファイル名の取得
```

```

25:     var fileName = parameter.getFileName()
26:
27:     // ファイルの書き出し
28:     var vf = new VirtualFile("filebox/" + fileName);
29:     var res = vf.save(fileData);
30:
31: }

```

**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る、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。



&lt;実行画面(結果)&gt;

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。



### 3.1.3 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) >
1: <HTML>
2:   <HEAD>
3:     <TITLE>File Center</TITLE>
4:   </HEAD>
5:   <BODY bgcolor="WhiteSmoke">
6:     <H2>Upload</H2>
7:     <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
8:       <INPUT type="file" name="local_file">
9:       <INPUT type="submit" value=" send ">
10:      </IMART>
11:      <HR>
12:      <H2>File List</H2>
13:      <TABLE>
14:        <TR><TH>File Name</TH><TH></TH></TR>
15:        <IMART type="repeat" list=fList item="rec">
16:          <TR>
17:            <TD><IMART type="string" value=rec></IMART></TD>
18:            <TD>
19:              <IMART type="link" action="action_download" server_file=rec>download</IMART>
20:            </TD>
21:          </TR>
22:        </IMART>
23:      </TABLE>
24:    </BODY>
25: </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>
1: var fList, rec;
2:
3: // 画面初期化
4: function init(request) {
5:   var root = new VirtualFile("filebox");
6:
7:   // 保存ディレクトリの有無のチェック
8:   if(! root.isDirectory()){
9:     root.makeDirectories();
10:   }
11:
12:   fList = root.files(); // ファイルリストの取得
13: }
14:
```

```

15: // 指定ファイルの受信
16: function action_upload(request) {
17:
18:     // パラメータ情報(=RequestParameter オブジェクト)を取得
19:     var parameter = request.getParameter("local_file");
20:
21:     // ファイルの中身を取得 (バイナリ)
22:     var fileData = parameter.getValueAsStream();
23:
24:     // ファイル名の取得
25:     var fileName = parameter.getFileName()
26:
27:     // ファイルの書き出し
28:     var vf = new VirtualFile("filebox/" + fileName);
29:     var res = vf.save(fileData);
30:
31: }
32:
33: // 指定ファイルの送信
34: function action_download(request) {
35:     var fpath = new VirtualFile("filebox/" + request.server_file); // 取得
36:     Module.download.send(fpath.load(), request.server_file); // 送信
37: }

```



&lt;実行画面(結果)&gt;



## Column

## 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。



### 3.1.4 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) >
1: <HTML>
2:   <HEAD>
3:     <TITLE>File Center</TITLE>
4:   </HEAD>
5:   <BODY bgcolor="WhiteSmoke">
6:     <H2>Upload</H2>
7:     <IMART type="form" action="action_upload" method="POST" enctype="multipart/form-data">
8:       <INPUT type="file" name="local_file">
9:       <INPUT type="submit" value=" send ">
10:    </IMART>
11:    <HR>
12:    <H2>File List</H2>
13:    <TABLE>
14:      <TR><TH>File Name</TH><TH></TH></TR>
15:      <IMART type="repeat" list=fList item="rec">
16:        <TR>
17:          <TD><IMART type="string" value=rec></IMART></TD>
18:          <TD>
19:            <IMART type="link" action="action_download" server_file=rec>download</IMART>
20:            / <IMART type="link" action="action_remove" server_file=rec>remove</IMART>
21:          </TD>
22:        </TR>
23:      </IMART>
24:    </TABLE>
25:  </BODY>
26:</HTML>
```

```
<削除用関数を追加したファンクション・コンテナ (filer.js) >
1: var fList, rec;
2:
3: // 画面初期化
4: function init(request) {
5:   var root = new VirtualFile("filebox");
6:
7:   // 保存ディレクトリの有無のチェック
8:   if(! root.isDirectory()){
9:     root.makeDirectories();
10:   }
11:
12:   fList = root.files(); // ファイルリストの取得
13: }
14:
15: // 指定ファイルの受信
16: function action_upload(request) {
17:
18:   // パラメータ情報 (=RequestParameter オブジェクト) を取得
19:   var parameter = request.getParameter("local_file");
20:
```

```

21: // ファイルの中身を取得 (バイナリ)
22: var fileData = parameter.getValueAsStream();
23:
24: // ファイル名の取得
25: var fileName = parameter.getFileName()
26:
27: // ファイルの書き出し
28: var vf = new VirtualFile("filebox/" + fileName);
29: var res = vf.save(fileData);
30:
31: }
32:
33: // 指定ファイルの送信
34: function action_download(request) {
35:     var fpath = new VirtualFile("filebox/" + request.server_file); // 取得
36:     Module.download.send(fpath.load(), request.server_file); // 送信
37: }
38:
39: // 指定ファイルの削除
40: function action_remove(request) {
41:     var fpath = new VirtualFile("filebox/" + request.server_file); // 取得
42:     fpath.remove(); // 削除
43: }

```



&lt;実行画面(結果)&gt;

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。



Column

## 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。



Column

## 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。

# 3.2

## 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。



### 3.2.1 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) >
1:  <HTML>
2:    <HEAD>
3:      <TITLE>Mail Sender</TITLE>
4:    </HEAD>
5:    <BODY bgcolor="WhiteSmoke">
6:      <IMART type="form" action="action_send" method="POST">
7:        To: <INPUT type="text" name="mail_to"><BR>
8:        From: <INPUT type="text" name="mail_from"><BR>
9:        Subject: <INPUT type="text" name="mail_subject"><BR>
10:       Message: <TEXTAREA name="mail_body" cols="40" rows="8"></TEXTAREA><BR>
11:       <INPUT type="submit" value=" send ">
12:     </IMART>
13:   </BODY>
14: </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) >
```

```
1: // メールの送信
2: function action_send(request) {
3:
4:   // ロケールの取得
5:   var locale = AccessSecurityManager.getSessionInfo().locale;
6:
7:   // MailSender オブジェクトを生成
8:   var mailSender = new MailSender(locale);
9:
10:  // 送信情報の設定
11:  mailSender.addTo(request.mail_to);
12:  mailSender.setFrom(request.mail_from);
13:  mailSender.setSubject(request.mail_subject);
14:
15:  // 本文の設定
16:  mailSender.setText(request.mail_body);
17:
18:  // メールの送信
19:  if( mailSender.send() ){
20:    // 送信成功
21:    Module.alert.reload("SYSTEM.SUCCESS",
22:                        "メールを送信しました。");
23:  }
24:  else{
```

```

25:      // 送信失敗
26:      Module.alert.reload("SYSTEM. ERR",
27:                            mailSender.getErrorMessage());
28:  }
29:
30: }

```



&lt;実行画面(結果)&gt;

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。

**i**  
Column

## 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。

**i**  
Column

## 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。

**i**  
Column

## 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。



### 3.2.2 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) >
1:  <HTML>
2:    <HEAD>
3:      <TITLE>Mail Sender</TITLE>
4:    </HEAD>
5:    <BODY bgcolor="WhiteSmoke">
6:      <IMART type="form" action="action_send" method="POST" enctype="multipart/form-data">
7:        To: <INPUT type="text" name="mail_to"><BR>
8:        From: <INPUT type="text" name="mail_from"><BR>
9:        Subject: <INPUT type="text" name="mail_subject"><BR>
10:       Attachment: <INPUT type="file" name="mail_file"><BR>
11:       Message: <TEXTAREA name="mail_body" cols="40" rows="8"></TEXTAREA><BR>
12:       <INPUT type="submit" value=" send ">
13:     </IMART>
14:   </BODY>
15: </HTML>
```



Column

### 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">

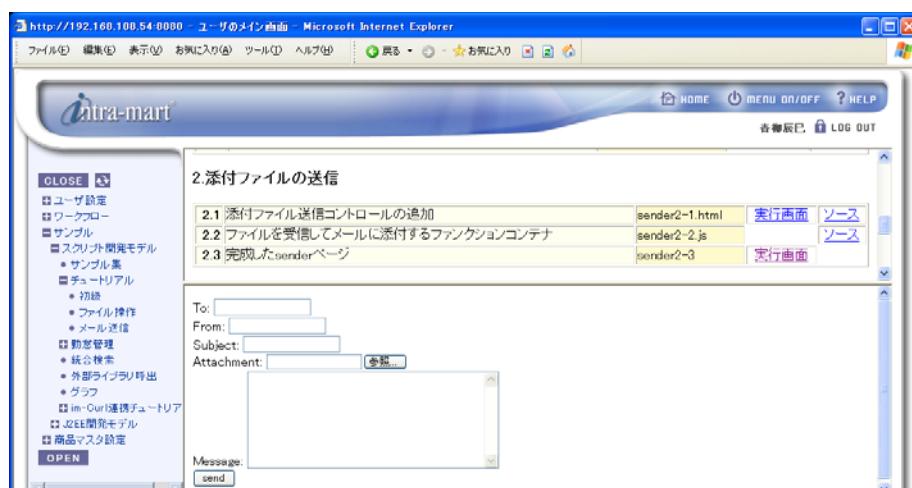
こうすることで、サーバ上では、フォーム・コントロール <INPUT type="file"> によりローカルのファイルを受け取ることができます。

```
<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) >
1: // メールの送信
2: function action_send(request) {
3:
4:   // ロケールの取得
5:   var locale = AccessSecurityManager.getSessionInfo().locale;
6:
7:   // MailSender オブジェクトを生成
8:   var mailSender = new MailSender(locale);
9:
10:  // 送信情報の設定
11:  mailSender.addTo(request.mail_to);
12:  mailSender.setFrom(request.mail_from);
13:  mailSender.setSubject(request.mail_subject);
14:
15:  // 添付ファイルの設定(RequestParameter オブジェクトとして取得)
16:  var parameter = request.getParameter("mail_file");
17:
18:  if( parameter != null && parameter.getLength() > 0 ) {
19:
20:    // ファイル名の取得
21:    var fileName = parameter.getFileName()
```

```

22:         // ファイルの中身を取得（バイナリ）
23:         var fileData = parameter.getValueAsStream();
24:
25:
26:         // 添付ファイルの設定
27:         mailSender.addAttachment(fileName, fileData);
28:
29:     }
30:
31:     // 本文の設定
32:     mailSender.setText(request.mail_body);
33:
34:     // メールの送信
35:     if( mailSender.send() ){
36:         // 送信成功
37:         Module.alert.reload("SYSTEM.SUCCESS",
38:                             "メールを送信しました。");
39:     }
40:     else{
41:         // 送信失敗
42:         Module.alert.reload("SYSTEM.ERR",
43:                             mailSender.getErrorMessage());
44:     }
45:
46: }

```



&lt;実行画面(結果)&gt;



## Column

## 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 II 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。



Column

## 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る場合があります。

### 3.3

## 拡張<IMART>タグ機能の使用例

拡張<IMART>タグとは、intra-mart側で既に用意されている<IMART>タグ以外に、独自の機能を持つ任意の<IMART>タグを定義できる機能です。

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。)



### 3.3.1 タグの定義と登録

ファンクション・コンテナに、拡張<IMART>タグの新しいtype属性を定義します。あわせて、そのtype属性に対応する実行関数を記述します。このファイルは、1度だけ実行されれば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(このことから、通常初期起動時に1度だけ実行されるようにします)。

```
1: // 関数と呼び出しキーワードの登録
2: // 呼び出しキーワードJP_DATEに対して、実行関数jupdate()を
3: // 登録します。
4: Imart.defineType("JP_DATE", jupdate);
5:
6: // 実行関数の定義
7: // 【引数】oAttr : タグ属性引数群オブジェクト
8: //          oInner: <IMART> および </IMART> に挟まれた部分
9: // 【返却】HTMLソース
10: // 【概要】<IMART type="JP_DATE"> の処理実行関数
11: // 属性 date に指定された日付データをフォーマット
12: // 変換して表示します。
13: function jupdate(oAttr, oInner){
14:     var target = oAttr.date;                      // 表示対象時間
15:     var format = "yyyy年MM月dd日 hh時mm分ss秒"; // 表示形式
16:     var src = Format.fromDate(format, target);   // 表示文字列
17:
18:     // <IMART type="JP_DATE"> と </IMART> に挟まれた部分の
19:     // プrezentation・ページ内ソースの解釈と実行および
20:     // 表示ソースの取得
21:     var sub = oInner.execute();
22:
23:     // 表示ソースの返却
24:     return src + sub;
25: }
```

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。



### 3.3.2 拡張<IMART>タグの利用

sample.html

使用するには、プレゼンテーション・ページ内で拡張<IMART>タグを記述します。

```
<IMART type="JP_DATE" date=now></IMART>
```

実行されると、ファンクション・コンテナからバインドされた日付オブジェクトの内容が表示されます。

sample.js

拡張<IMART>タグに対してバインドする日付データを作成します。

```
var now; // バインド変数
function init(request){
    now = new Date(); // 現在の時間情報の取得
}
```



- 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。



Column

#### 拡張<IMART>タグについて

<IMART type="XXX">の“XXX”部分をユーザ独自に定義することができます。詳細は、「APIリスト」の画面共通モジュールの拡張<IMART>タグ項を参照してください。



Column

#### <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)
```

# 3.4 ユーザ定義関数の登録と利用

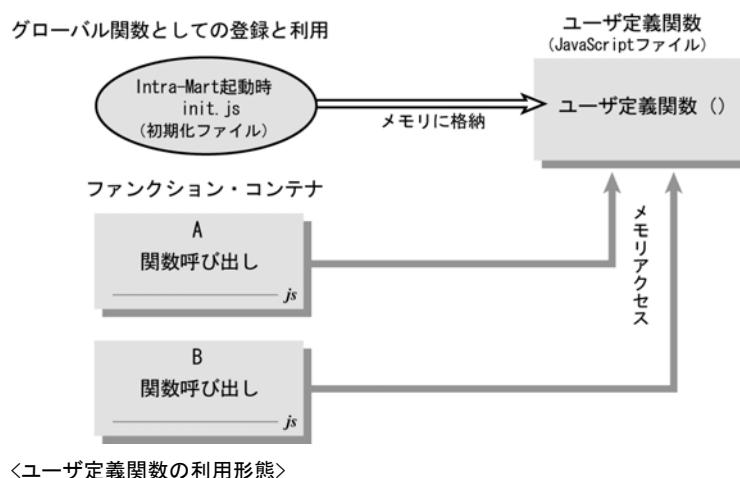
intra-mart WebPlatformを利用した開発では、JavaScriptで記述したユーザ定義関数を「グローバル関数」として登録する方法があります。

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。

以下に、ユーザ定義関数をグローバル関数として、登録・利用する方法について説明を行います。



- 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。



## 3.4.1 グローバル関数としての登録・呼び出し

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。

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、「library/common.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。

```
<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。

```
<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("library/common");
```



- 必ずしもpages/src/init.js内に記述しなくとも、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します。

ここでは例として、ファンクション・コンテナの「pages/src/applicationPath/app001.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。

```
<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>
```

```
/* applicationPath/app001.js */

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

# 3.5

# 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。



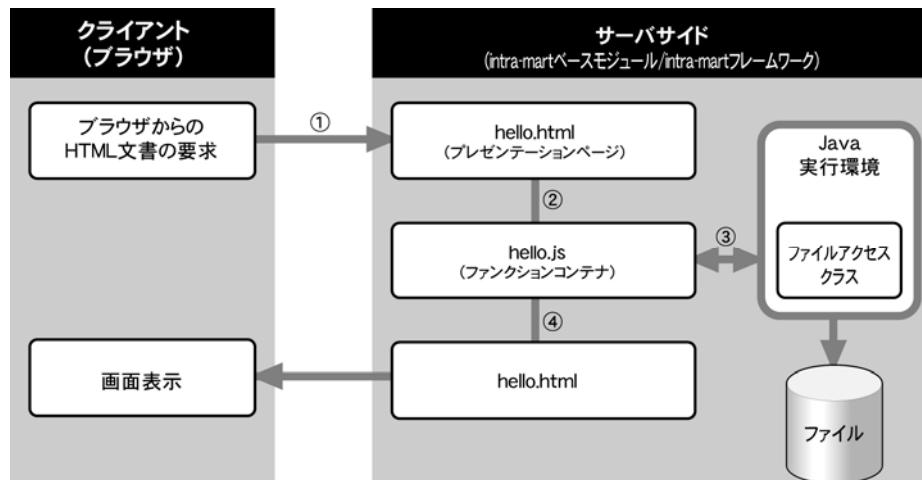
## 3.5.1 標準JavaClassとの連携方法

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。

**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み込みサーバサイド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)

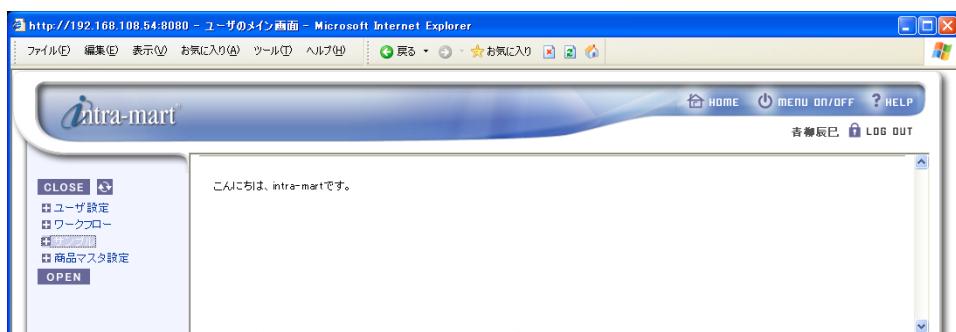
    //Javaクラス DataInputStreamをJavaScriptオブジェクトとして生成
    var javaObjDataIn = new java.io.DataInputStream(javaObjFileIn); (2)

    //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。



&lt;hello.htmlの実行画面&gt;



Column

## 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。



### 3.5.1.1 標準JavaClass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。



### 3.5.1.2 自作JavaClassとの連携方法

自作JavaClassとの連携を行う場合、intra-mart起動時の設定と自作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。



### 3.5.1.3 intra-mart起動時の設定方法

intra-mart起動時のjavaコマンドの「-cp」オプションで、自作JavaClassのあるディレクトリまたはjarファイルをクラスパスとして追加設定しなければなりません。javaコマンドの「-cp」オプションの使い方についてはJavaのリファレンスを参照してください。



- クラスパスは、「-cp」オプションで指定します。設定はconf/imart.xmlで行います。



### 3.5.1.4 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。
```



### 3.5.1.5 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( request ) {

    //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み込み内容を渡す。
        nameValue = Unicode.from(javaObjString);
    }
}
```

```

<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() {
    }

    //ファイル読み込みメソッド
    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 );

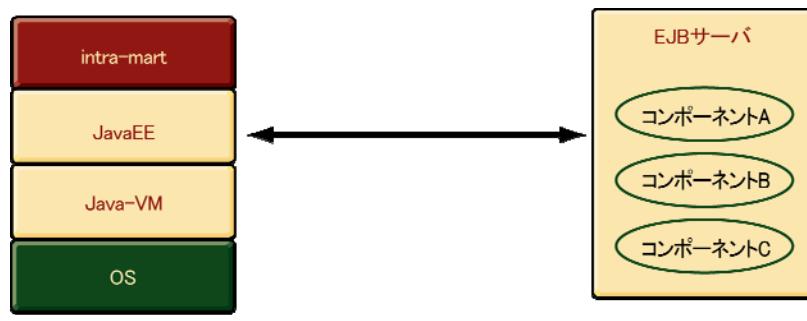
            //ファイルを1レコード読み込み
            readstr = ds.readLine();

            //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;
    }
}

```

# 3.6 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〉



## 3.6.1 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。



## 3.6.2 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.java.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。

## 3.7

# 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。

### ■ 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
```



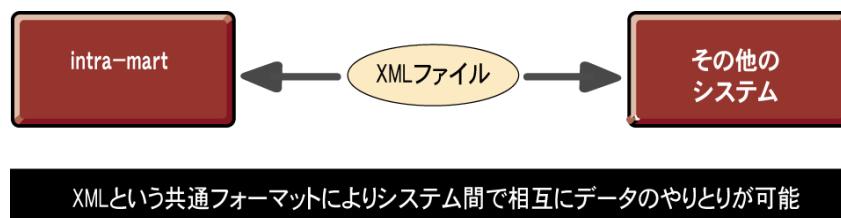
- プロセスの終了コードは、0の場合正常終了となります。
- 詳細は、APIリスト「アプリケーション共通モジュール」のグローバル関数「execute()」を参照してください。

## 3.8

# 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なうことができます。



### 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ことができます。



- 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

```

1 : function init(request) {
2 :     var userId      = request.getParameterValue("/account/user-id");
3 :     var name       = request.getParameterValue("/account/name");
4 :     var roleId      = request.getParameterValue("/account/role/role-id");
5 :     var sampleAttr = request.getParameterValue("/account/role/role-id/@sample-attr");
6 :
7 :     Debug.browse(userId, name, roleId, sampleAttr);
8 : }
```



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ーフィールドが「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

```

1 : function init(request) {
2 :
3 :     //-----
4 :     // メッセージボディを取得
5 :     //-----
6 :     var messageBody = request.getMessageBody("UTF-8");
7 :
8 :     //-----
9 :     // XMLデータの構文解析
10 :    //-----
11 :    var xmlParser = new XMLParser();
12 :    var doc = xmlParser.parseString(messageBody);
13 :
14 :    if(xmlParser.isError()){
15 :        Debug.browse("エラーが発生しました。", xmlParser.getErrorMessage());
16 :    }
17 :
18 :    //-----
19 :    // <user-id>, <name>, <role>ノード取得
20 :    //-----
21 :    var childNodes = null;
22 :    var accountNode = doc.getDocumentElement();
23 :    var userIdNode = null;
24 :    var nameNode   = null;
```

```

25 :     var roleNode    = null;
26 :
27 :     childNodes = accountNode.getChildNodes();
28 :     for(var i = 0 ; i < childNodes.length ;i++) {
29 :         if(childNodes[i].getTagName() == "user-id") {
30 :             userIdNode = childNodes[i];
31 :         }
32 :         else if(childNodes[i].getTagName() == "name") {
33 :             nameNode = childNodes[i];
34 :         }
35 :         else if(childNodes[i].getTagName() == "role") {
36 :             roleNode = childNodes[i];
37 :         }
38 :     }
39 :
40 :     //-----
41 :     // <role-id>ノード取得
42 :     //-----
43 :     var roleIdNode = null;
44 :     childNodes = roleNode.getChildNodes();
45 :     for(var i = 0 ; i < childNodes.length ;i++) {
46 :         if(childNodes[i].getTagName() == "role-id") {
47 :             roleIdNode = childNodes[i];
48 :         }
49 :     }
50 :
51 :     //-----
52 :     // <role-id>ノードの属性取得
53 :     //-----
54 :     var roleIdAttr = roleIdNode.getAttribute("sample-attr");
55 :
56 :     //-----
57 :     // 各ノードの値を表示
58 :     //-----
59 :     Debug.browse(userIdNode.getChildNodes()[0].getValue(),
60 :                  nameNode.getChildNodes()[0].getValue(),
61 :                  roleIdNode.getChildNodes()[0].getValue(),
62 :                  roleIdAttr);
63 :
64 : }
```



### 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rezentation・ページ(.html)の作成

```

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

#### ■ ファンクション・コンテナ(js)の作成

```

1 : var xmlString = "";
2 :
3 : function init(request) {
4 :
5 :   //-----
6 :   // DOMツリーを構築
7 :   //-----
8 :   var doc = new XMLDocument("<account/>");
9 :   var accountNode = doc.getDocumentElement();
10 :
11 :  // エレメントを作成
12 :  var userIdNode = doc.createElement("user-id");
13 :  var nameNode = doc.createElement("name");
14 :  var roleNode = doc.createElement("role");
15 :  var roleIdNode = doc.createElement("role-id");
16 :
17 :  // テキストノードを作成
18 :  var userIdText = doc.createTextNode("ueda");
19 :  var nameText = doc.createTextNode("上田");
20 :  var roleIdText = doc.createTextNode("level1");
21 :
22 :  // 属性を設定
23 :  roleIdNode.setAttribute("sample-attr", "サンプル属性");
24 :
25 :  // 子ノードを追加
26 :  userIdNode.appendChild(userIdText);
27 :  nameNode.appendChild(nameText);
28 :  roleNode.appendChild(roleIdNode);
29 :  roleIdNode.appendChild(roleIdText);
30 :
31 :  accountNode.appendChild(userIdNode);
32 :  accountNode.appendChild(nameNode);
33 :  accountNode.appendChild(roleNode);
34 :
35 :  //-----
36 :  // XMLの文字列をバインド
37 :  //-----
38 :  xmlString = doc.getXmlString();
39 :
40 : }
```



### 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

```

1 : function init(request) {
2 :
3 :     //-----
4 :     // DOMツリーを構築
5 :     //-----
6 :     var doc = new XMLDocument("<account/>");
7 :     var accountNode = doc.getDocumentElement();
8 :
9 :     // エレメントを作成
10 :    var userIdNode = doc.createElement("user-id");
11 :    var nameNode = doc.createElement("name");
12 :    var roleNode = doc.createElement("role");
13 :    var roleIdNode = doc.createElement("role-id");
14 :
15 :    // テキストノードを作成
16 :    var userIdText = doc.createTextNode("ueda");
17 :    var nameText = doc.createTextNode("上田");
18 :    var roleIdText = doc.createTextNode("level1");
19 :
20 :    // 属性を設定
21 :    roleIdNode.setAttribute("sample-attr", "サンプル属性");
22 :
23 :    // 子ノードを追加
24 :    userIdNode.appendChild(userIdText);
25 :    nameNode.appendChild(nameText);
26 :    roleNode.appendChild(roleIdNode);
27 :    roleIdNode.appendChild(roleIdText);
28 :
29 :    accountNode.appendChild(userIdNode);
30 :    accountNode.appendChild(nameNode);
31 :    accountNode.appendChild(roleNode);
32 :
33 :    //-----
34 :    // XMLの文字列をバインド
35 :    //-----
36 :    var encoding = "UTF-8";
37 :    var xmlString = "<?xml version='1.0' encoding=" + encoding + "?>" + doc.xmlString();
38 :
39 :    //-----
40 :    // Content-Typeを設定
41 :    //-----
42 :    var response = Web.getHTTPResponse();
43 :    response.setContentType("text/xml; charset=" + encoding);
44 :
45 :    //-----
46 :    // データ送信
47 :    //-----
48 :    response.sendMessageBodyString(xmlString);
49 :
50 : }
```

# 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できます。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、E4X では、XML の初期化に ◇ を使うことができます。

```
1: var xml = <root>
2:           <node attr="0">サンプル</node>
3:         </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。

```
1: var src = "<root><node attr='0'>サンプル</node></root>";
2: 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。

```
1: var xml = <root>
2:           <node attr="0">サンプル0</node>
3:         </root>;
4:
5: Debug.print(xml.node);           // 要素の値「サンプル0」
6: Debug.print(xml.node.@attr);    // 属性の値「0」
7: 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できます。
```



### 3.9.4 子ノードの取得

「`children()`」や「`.*`」を利用して子ノードを取得することができます。

```

1: var xml = <root>
2:           <node1>AAAA</node1>
3:           <node1>BBBB</node1>
4:           <node2 num="0">CCCC</node2>
5:           <node2 num="1">
6:             <node2Child>DDDD</node2Child>
7:           </node2>
8:         </root>;
9:
10: var children = xml.children(); // 「var children = xml.*;」と記述することも可能
11:
12: for(var prop in children){
13:   Debug.print("children[" + prop + "] = " + children[prop]);
14: }
15:
16: // for each 文を利用して値を取得
17: for each (var value in children){
18:   Debug.print("value = " + value);
19: }
```



### 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。

```

1: var xml = <root>
2:           <node1>AAAA</node1>
3:         </root>;
4:
5: Debug.print("追加前:" + xml.toString());
6:
7: xml.addNode      = "BBBB"; // 子ノードの追加
8: xml.addNode.@id = "CCCC"; // 属性の追加
9:
10: 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。

```

1: var xml = <root>
2:           <node1>AAAA</node1>
3:           <node1>BBBB</node1>
4:           <node2 num="0">CCCC</node2>
5:           <node2 num="1">
6:             <node2Child>DDDD</node2Child>
7:           </node2>
8:         </root>;
9:
10: Debug.print("削除前:" + xml.toString());
11:
```

```

12: delete xml.node1[1];
13: delete xml.node2[1].@num;
14:
15: 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。

```

1: var attrName = "code";
2: var tagName = "name";
3:
4: var attrVal = "0001";
5: var content = "商品1";
6:
7: var xml = <order>
8:           <item {attrName}={attrVal}>
9:             <{tagName}>{content}</{tagName}>
10:            </item>
11:          </order>;
12:
13: Debug.print("=====");
14: Debug.print(xml.toXMLString());
15: Debug.print("=====");
16:
17:
18: // 「商品2」を追加
19: attrVal = "0002";
20: content = "商品2";
21: xml.appendChild(
22:           <item {attrName}={attrVal}>
23:             <{tagName}>{content}</{tagName}>
24:           </item>
25: );
26:
27: // 「商品3」を追加
28: attrVal = "0003";
29: content = "商品3";
30: xml.appendChild(
31:           <item {attrName}={attrVal}>
32:             <{tagName}>{content}</{tagName}>
33:           </item>
34: );
35: Debug.print("=====");
36: Debug.print(xml.toXMLString());
37: Debug.print("=====");
38:
39:
40: // 「商品2」を削除
41: delete xml.item[1];
42:
43: Debug.print("=====");
44: Debug.print(xml.toXMLString());
45: Debug.print("=====");

```

# 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 `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ついて

<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

```

1: <html>
2:   <head>
3:     <IMART type="jsspRpc"
4:       name="jsSample"
5:       page="jssp_rpc_test/sample1">
6:   </IMART>
7:
8:   <script language="JavaScript">
9:     /**
10:      * 「jssp_rpc_test/sample1.js」の関数「getNow()」を実行します。
11:      */
12:     function execute() {
13:       try{
14:         var result = jsSample.getNow("Now = ");
15:         alert(result);
16:       }
17:       catch(ex) {
18:         alert(ex.message);
19:         return;
20:       }
21:     }
22:   </script>
23:   <head>
24:
25:   <body>
26:     <input type="button" value="実行（同期）" onclick="execute();">
27:   </body>
28: </html>

```



#### 3.10.3.2 サーバサイドのJSソース「`jssp_rpc_test/sample1.js`」

```

1: function getNow( args ){
2:   return args + (new Date()).toString();
3: }

```



## 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

```

1: <html>
2:   <head>
3:     <IMART type="jsspRpc"
4:       name="jsSample"
5:       page="jssp_rpc_test/sample2"
6:       callback="callbackFunction">
7:   </IMART>
8:
9:   <script language="JavaScript">
10:    /**
11:     * 「jssp_rpc_test/sample2.js」の関数「getObject()」を実行します。
12:    */
13:    function execute() {
14:      // 引数作成
15:      var obj = new Object();
16:      obj.stringProp = "value1";
17:      obj.booleanProp = true;
18:      obj.numberProp = -15;
19:      obj.arrayProp = new Array();
20:      obj.arrayProp[0] = "ary0";
21:      obj.arrayProp[1] = "ary1";
22:      obj.arrayProp[2] = "ary2";
23:      obj.dateProp = new Date();
24:
25:      // 内容を確認
26:      var str = "";
27:      str += "コールバック関数を確認するために" + "\n";
28:      str += "サーバサイドで5秒間スリープします。" + "\n";
29:      str += "-----" + "\n";
30:      str += "実行前" + "\n";
31:      str += "-----" + "\n";
32:      str += "obj.stringProp = " + obj.stringProp + "\n";
33:      str += "obj.booleanProp = " + obj.booleanProp + "\n";
34:      str += "obj.numberProp = " + obj.numberProp + "\n";
35:      str += "obj.arrayProp[0] = " + obj.arrayProp[0] + "\n";
36:      str += "obj.arrayProp[1] = " + obj.arrayProp[1] + "\n";
37:      str += "obj.arrayProp[2] = " + obj.arrayProp[2] + "\n";
38:      str += "obj.dateProp = " + obj.dateProp + "\n";
39:      str += "-----" + "\n";
40:
41:      alert(str);
42:
43:      // サーバロジック実行
44:      jsSample.getObject(obj);
45:    }
46:
47:
48:    /**

```

```

49:         * コールバック関数
50:         */
51:         function callBackFunction( result ){
52:
53:             // 内容を確認
54:             var str = "";
55:             str += "実行後" + "\n";
56:             str += "-----" + "\n";
57:             str += "result.stringProp = " + result.stringProp + "\n";
58:             str += "result.booleanProp = " + result.booleanProp + "\n";
59:             str += "result.numberProp = " + result.numberProp + "\n";
60:             str += "result.arrayProp[0] = " + result.arrayProp[0] + "\n";
61:             str += "result.arrayProp[1] = " + result.arrayProp[1] + "\n";
62:             str += "result.arrayProp[2] = " + result.arrayProp[2] + "\n";
63:             str += "result.dateProp = " + result.dateProp + "\n";
64:             str += "-----" + "\n";
65:
66:             alert(str);
67:         }
68:
69:     </script>
70:     <head>
71:
72:     <body>
73:         <input type="button" value="実行（非同期）" onclick="execute();"
74:     </body>
75: </html>

```



#### 3.10.4.2 サーバサイドのJSソース「jssp\_rpc\_test/sample2.js」

```

1: function getObject( args ){
2:
3:     // 受け取ったオブジェクトの内容を表示
4:     for(var prop in args){
5:         if (args.hasOwnProperty(prop)) {
6:             Debug.print(prop + ":" + args[prop] + "[" + typeof args[prop] + "]")
7:         }
8:     }
9:
10:    // 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）
11:    Client.sleep(5 * 1000);
12:
13:    // 受け取ったオブジェクトの内容を加工
14:    args.stringProp = args.stringProp + " (modified !)";
15:    args.booleanProp = false;
16:    args.numberProp = args.numberProp + 10000;
17:    args.arrayProp[0] = args.arrayProp[0] + " (modified !)";
18:    args.arrayProp[1] = args.arrayProp[1] + " (modified !)";
19:    args.arrayProp[2] = args.arrayProp[2] + " (modified !)";
20:    args.dateProp.setFullYear(2100);
21:    args.dateProp = args.dateProp;
22:
23:    // 結果を返却
24:    return args;
25: }

```

# 3.11

## 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。



### 3.11.1 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
    }
```

The screenshot shows the 'Debug for Script variables' window in a browser. The window title is 'Debug for Script variables.' It displays a hierarchical tree of variables under the 'data' category. The tree has six levels of nesting, with each node having a 'staff\_id' and four 'stf\_name\_\*' properties. The values for 'staff\_id' range from 11001 to 11006. The values for 'stf\_name\_\*' include 'シヤイ-1', 'シヤイ-1', 'シヤイ-2', 'シヤイ-2', 'シヤイ-3', 'シヤイ-3', 'シヤイ-4', 'シヤイ-4', 'シヤイ-5', 'シヤイ-5', and 'シヤイ-6'. The right side of the window shows the variable types: 'Number' for 'staff\_id' and 'String' for 'stf\_name\_\*'.

<デバッグ結果表示画面>



## 3.11.2 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>
1: //=====
2: //      【入力】request: URL引数取得オブジェクト
3: //      【返却】なし
4: //      【概要】
5: //=====
6: function init(request){
7:     var now = new Date();
8:     Debug.print("デバッグ画面表示前");    // コンソールに出力
9:     Debug.browse(now);                    // 画面に出力
10:    Debug.print("デバッグ画面表示後");   // コンソールに出力
11: }
```

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。

```
サンプルアプリケーション - java -Xms16m intraAppGrv
Session Expire: bwoy1wpoz01 -> Wed Oct 04 12:07:47 JST 2000
Remove Session: bwoy1wpoz01
Compile Script: batch_is
Wed Oct 04 12:20:07 GMT+0900 (JST) 2000: test
Session Expire: bwoy20n17e02 -> Wed Oct 04 12:30:07 JST 2000
Remove Session: bwoy20n17e02
Wed Oct 04 13:20:07 GMT+0900 (JST) 2000: test
Session Expire: bwoy281dwo3 -> Wed Oct 04 13:30:07 JST 2000
Remove Session: bwoy281dwo3
Wed Oct 04 14:20:07 GMT+0900 (JST) 2000: test
Session Expire: bwoy2cim704 -> Wed Oct 04 14:30:03 JST 2000
Remove Session: bwoy2cim704
Session Expire: bwoy1wpoz01 -> Wed Oct 04 15:12:33 JST 2000
Return to Page: system/security/interface/main_frame
Return to Page: homepage
Return to Page: system/security/interface/menu
Return to Page: developer/main
Return to Page: developer/test/main
Return to Page: developer/test/explain
Compile Layout: developer/test/menu.html
Return to Page: developer/test/menu
Compile Script: developer/test/alanac/main.js
デバッグ画面表示前
Return to Page: developer/test/alanac/main
```

<DOSコンソール実行画面(結果)-1>

intra-mart[Debug.browse0] - Netscape

Debug for Script variables.

Wed Oct 04 15:09:14 JST 2000 [970639754133]

Date

**Bold:** Object property name or Array index  
**Italic:** Variable type  
**Normal:** Value

<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。



Column

## 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。



Column

## 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。

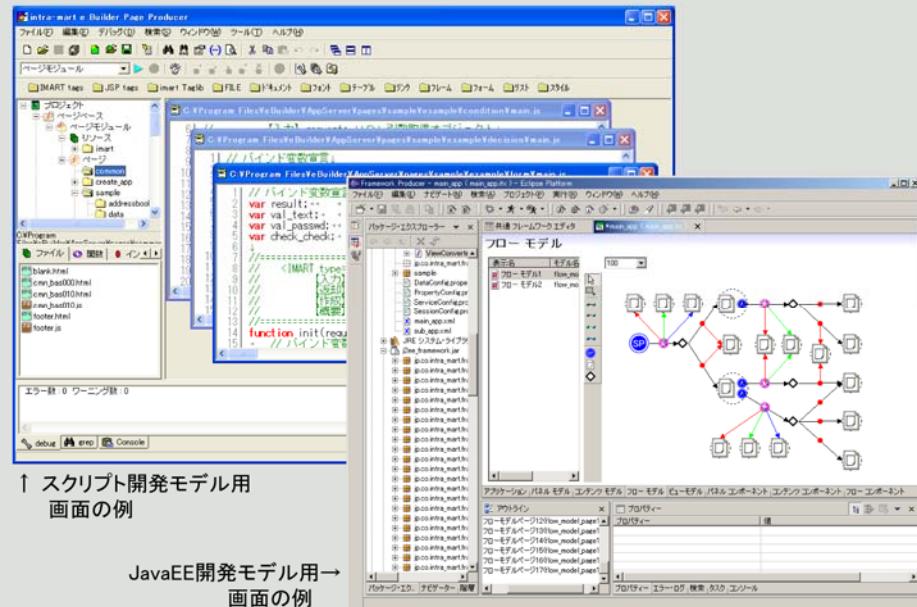


Column

## プログラム開発環境をサポートする「eBuilder 6.1」(2007/09/07リリース予定)

別売のintra-mart 「eBuilder Ver6.1」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 Ver6.1」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「eBuilder Framework Producer」の2種類が用意されています。

詳細は、チュートリアルガイドの「1.8 intra-mart eBuilder Ver6.1」を参照してください。



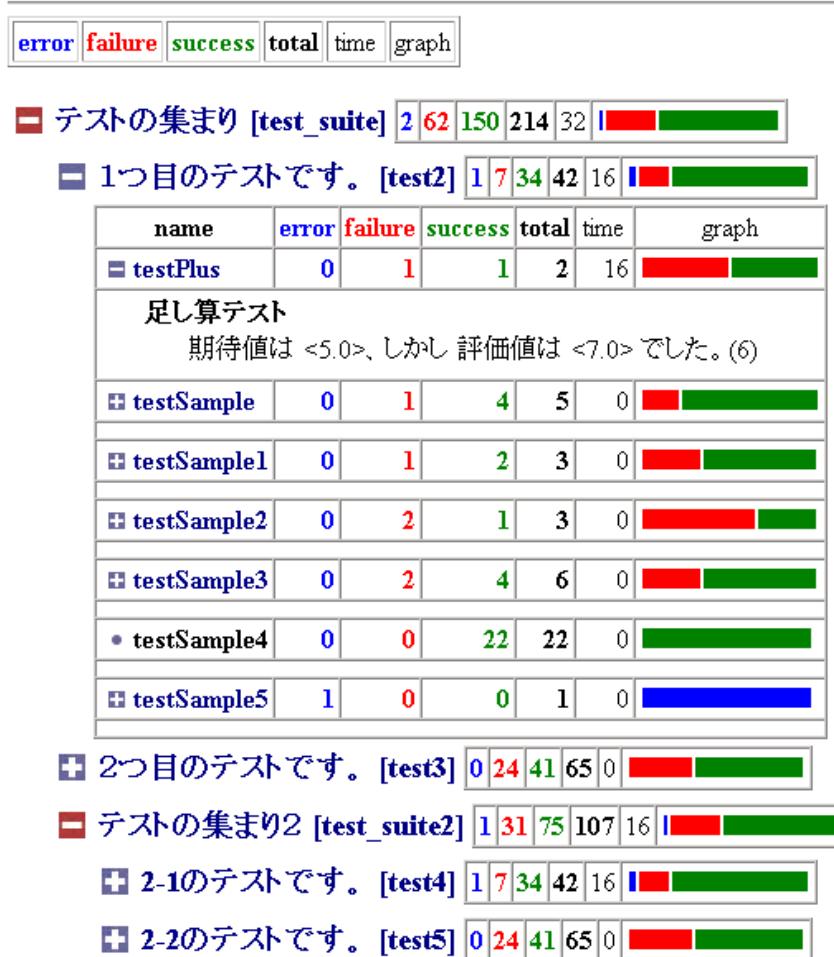
im-JsUnitは、スクリプト開発モデルにおける単体テスト環境を提供します。

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。

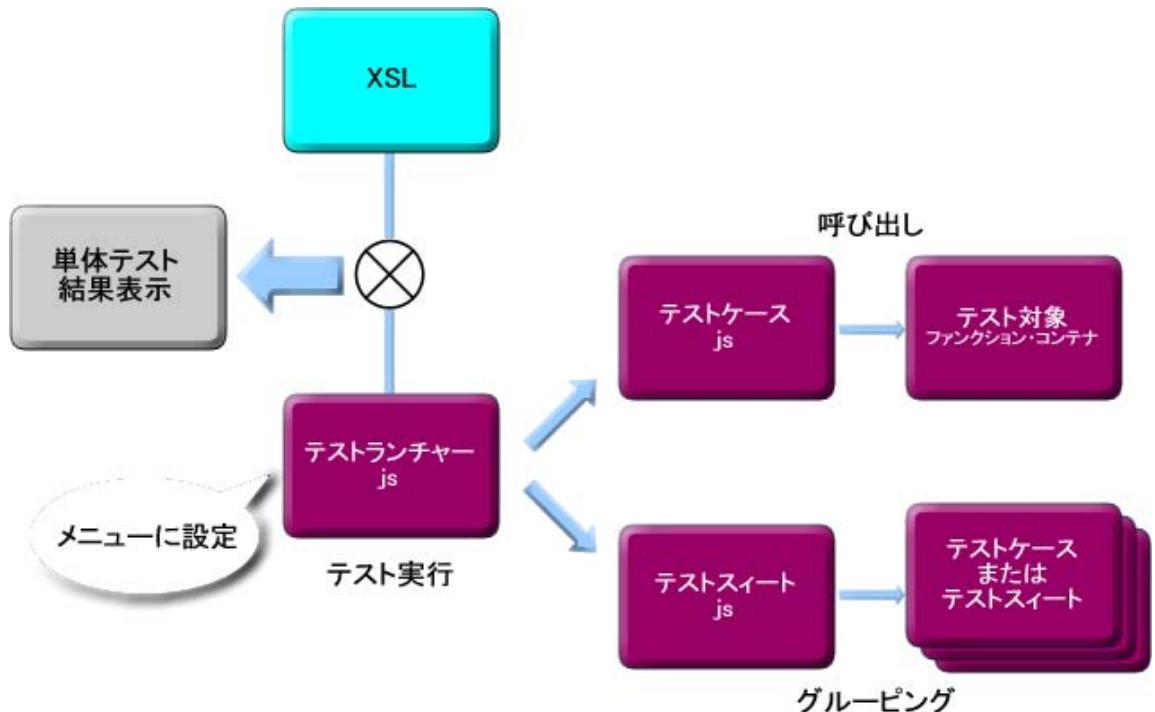
## im-JsUnit Result





### 3.12.1 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イ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。



#### 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。

#### テストスイート

テストスイートは、複数のテストケースおよびテストスイートを1つのテストグループにするものです。  
テストスイ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イートを一度に実行することができます。  
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イ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。



## 3.12.2 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。

<b>testXXXXXX()</b>	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。
<b>setUp()</b>	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。
<b>tearDown()</b>	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。
<b>oneTimeSetUp()</b>	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。
<b>oneTimeTearDown()</b>	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。
<b>defineTestSuite()</b>	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イートとして扱います。 その他の関数は無視されます。 テストスイ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。



### 3.12.3 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。
    var result = module.calcPlus(1, 2);
}
```

#### ■ テストスイートの作成

テストスイートの作成は、**defineTestSuite**関数を定義します。

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

**defineTestSuite**関数内でテストスイ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イートファイルを追加します。

```
function defineTestSuite() {

    // テストスイ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（テストスイート）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");

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



### 3.12.3.1 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

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



### 3.12.3.2 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。



### 3.12.4 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シ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シ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");

    // コンテンツタイプの定義
    // 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.sendResponseBody(result);
}
```



### 3.12.5 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シートファイル(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シ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。

# 3.13 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APIリスト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。



Column

## 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-31JK/charset</charset>
    <javascript>
        <compiler enable="true" />
        <!-- enable:true = Auto compiler to Java class -->
        <!-- enable:false = Interpreter -->

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

● **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。

● **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。

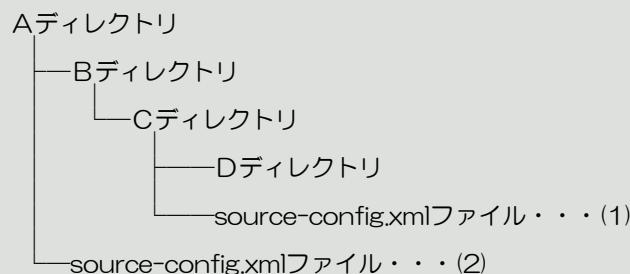
● **resource-file/view/compiler**

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。

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

- \* A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。
- \* B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。
- \* Cディレクトリ直下のプログラム：(1)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：(1)の設定内容が有効になります。



〈 source-config.xmlの配置例 〉



## 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。

```
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rezentation・ページの自動コンパイル設定
```

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
```



### 3.13.1 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。



Column

## %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(インターネット・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)
```



### 3.13.2 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ふことがあります。



- 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。

❖ パ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。



### 3.13.3 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()
```



### 3.13.4 制約



#### 3.13.4.1 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。



#### 3.13.4.2 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。
```

■ 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();
}

```

# 3.14 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
```

## ■ 使用例

```
// 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。

intra-mart WebPlatform/AppFrameworkには、スクリプト開発モデル用のサンプルとして「勤怠管理」が用意されています。



### 3.15.1 データベースへのサンプルデータの登録

登録したデータベースにサンプルのデータを登録します。サンプルデータのインポートは、システム管理者権限でログインし、[ライセンス] メニューで行います。プルダウンメニューで、サンプルデータをインポートするログイングループ名を選択して、[インポート] ボタンをクリックします。

The screenshot shows a Microsoft Internet Explorer window with the URL <http://pdg.intra-mart.jp>. The title bar says 'システム管理者のメイン画面 - Microsoft Internet Explorer'. The left sidebar has a tree view with nodes like 'システム環境', 'システム管理者設定', 'ライセンス', etc. The main content area is titled 'ライセンス' and shows a table for 'システムライセンス'. At the bottom right of this table, there is a button labeled 'Import'. Below the table, there is another row with a dropdown menu set to 'default' and a button labeled 'Import'. This second row is highlighted with a red rectangle.

<サンプル・アプリケーション用データのインポート>



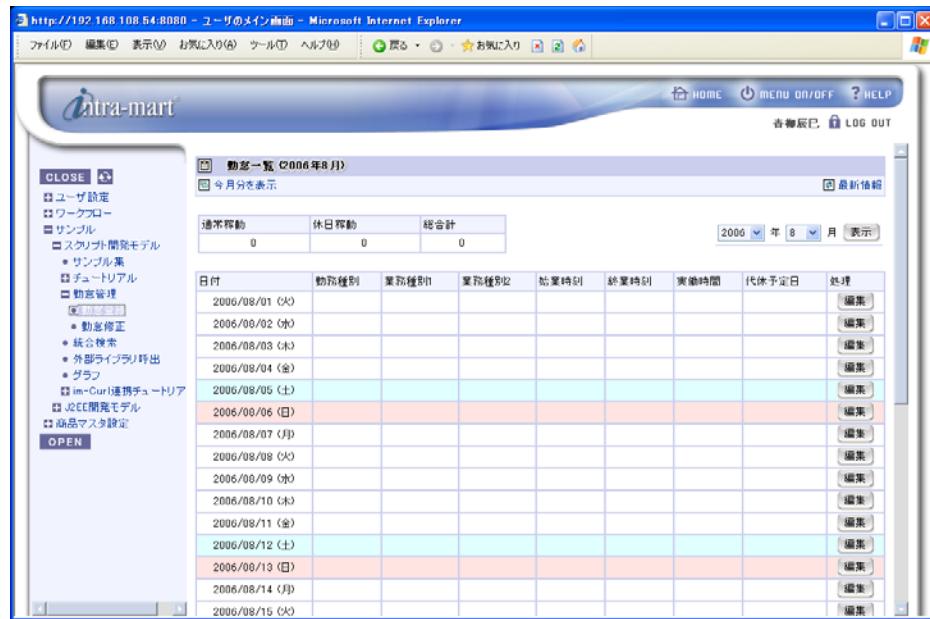
### 3.15.2 「勤怠管理」アプリケーションの操作

「勤怠管理」アプリケーションは、[サンプル] - [スクリプト開発モデル] - [勤怠管理] に [勤怠登録] と [勤怠修正] の2つのメニューが用意されています。



#### 3.15.2.1 勤怠登録

[サンプル] - [スクリプト開発モデル] - [勤怠管理] から [勤怠登録] を選択すると、 今月の勤怠情報が表示されます。別の月の勤怠情報を表示するには、画面右上のコンボボックスで表示したい年月を選択し、表示ボタンを押してください。勤怠を登録するには、登録したい日付の編集ボタンをクリックします。指定日に関する「勤務種別」・「業務種別」・「始業・終業時刻」等の勤怠内容を登録します。



&lt;勤怠一覧画面&gt;



&lt;勤怠登録画面&gt;



### 3.15.2.2 勤怠修正

勤怠年月および申請者のユーザコードをキーに検索し、申請された勤怠内容を呼び出し、修正することができます。検索条件入力画面にて、勤怠年月および申請者のユーザコードを入力し、検索ボタンを押すと、検索条件に一致した勤怠内容を表示します。勤怠を修正するには、修正したい日付の編集ボタンをクリックします。指定日に関する「勤務種別」・「業務種別」・「始業・終業時刻」等の勤怠内容を修正します。

This screenshot shows the 'Search Conditions Input' page (検索条件入力画面) of the intra-mart application. The URL is <http://192.168.108.54:8080>. The page includes a sidebar with navigation links like 'User Settings', 'Workflow', 'Sample', 'Attendance Management', 'Attendance Correction', etc. The main form has fields for 'Attendance Month' (勤怠年月) set to '2006 年 8 月' and 'User Code' (対象ユーザコード) set to 'aoyagi'. A 'Search' button (検索) is present.

&lt;検索条件入力画面&gt;

This screenshot shows the 'User Search' page (ユーザ検索画面) of the intra-mart application. The URL is <http://192.168.108.54:8080>. It displays a list of users found based on the search criteria: 'Search Date: [2006/08/10]', 'Category: [あ]', and 'Group: [会社・組織]'. The results show four users: aoyagi, ikuta, ohiso, and ueda. The 'aoyagi' entry is selected. A 'Decision' button (決定) is visible at the bottom.

&lt;ユーザ検索画面&gt;

This screenshot shows the 'Search Results' page (検索結果画面) of the intra-mart application. The URL is <http://192.168.108.54:8080>. It displays the search results for user 'aoyagi' from August 2006. The results table shows the following data:

日付	勤務種別	業務種別	業務種別2	始業時刻	終業時刻	実働時間	代休予定日	処理
2006/08/01 (火)								<input type="button" value="編集"/>
2006/08/02 (水)								<input type="button" value="編集"/>
2006/08/03 (木)								<input type="button" value="編集"/>
2006/08/04 (金)								<input type="button" value="編集"/>
2006/08/05 (土)								<input type="button" value="編集"/>
2006/08/06 (日)								<input type="button" value="編集"/>

&lt;検索結果画面&gt;



### 3.15.3 ワークフロー・モジュールとの連携

ワークフロー・モジュールを使用することで、簡単にワークフローに対応したアプリケーションを作成することができます。ワークフロー・モジュールとの連携のサンプル・アプリケーションが「%ResourceService%/pages/src/sample/bpw/purchase/standard」に収録されておりますのでご参照ください。

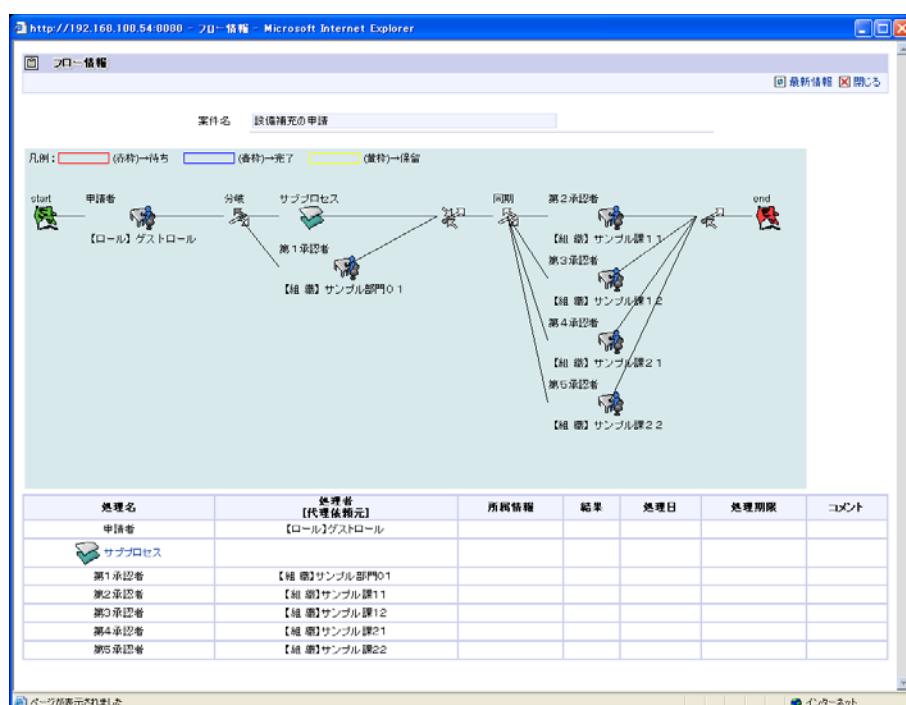


#### 3.15.3.1 申請画面

<ワークフローサンプル 起票>



#### 3.15.3.2 フロー情報画面



<ワークフローサンプル フロー情報>



### 3.15.3.3 承認画面

&lt;ワークフローサンプル 承認&gt;

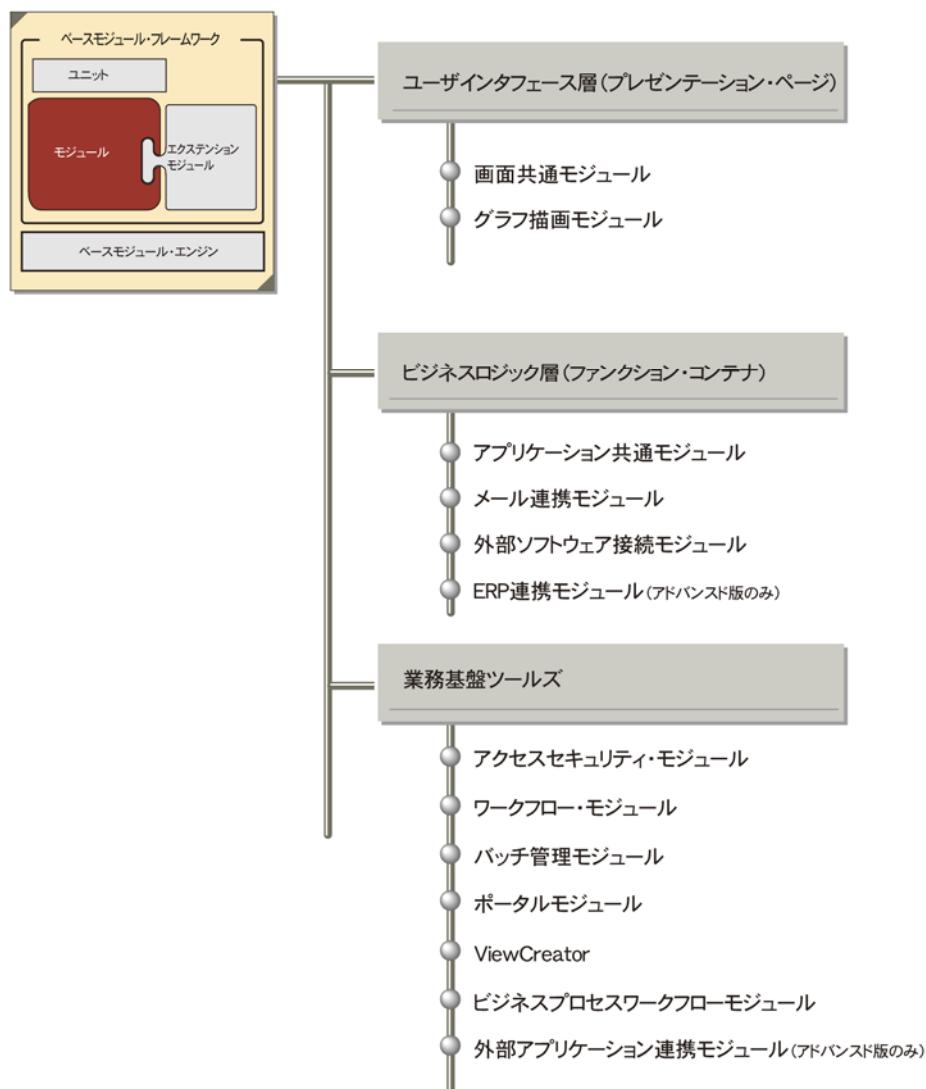


- サンプルのソースは、以下のフォルダに用意されています。  
%Resource Service%/pages/src/sample/bpw/purchase/standard
- intra-martでは、ワークフロー・モジュールを使用することで、簡単にワークフローに対応したアプリケーションを作成することができます。ワークフロー・モジュールとの連携やプロセス(承認ルート)の作成等のワークフロー機能の詳細は、別冊の「ワークフローガイド」「ワークフロー仕様書」を参照してください。

# 3.16

# モジュールの組み込みと操作

intra-martは前述のように「ユーザインターフェース層（プレゼンテーション・ページ）」、「ビジネスロジック層（ファンクション・コンテナ）」、「業務基盤ツール」に分類されます。ここでは各モジュールについて説明を行います。





## 3.16.1 ユーザインターフェース層

ユーザインターフェース層に属するモジュールを説明します。



### 3.16.1.1 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ー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

ユーザインター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群は、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ユーザインターフェースの例

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ー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ー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。

氏名(カナ)	スタッフ_ナマエ_カナ_000000	生年月日	1971 年 2 月 28 日
氏名(漢字)	スタッフ_名前_漢字_0000000	実年齢	26.92歳
氏名(英字)	Staff_Name_English_00000002	標準年齢	29.83歳
		満年齢	28.17歳
性別	<input checked="" type="radio"/> 男性 <input type="radio"/> 女性	入社年月日	1990 年 4 月 1 日
血液型	<input checked="" type="radio"/> A <input type="radio"/> B <input type="radio"/> O <input type="radio"/> AB <input type="radio"/> 不明	勤続年数	7.833333333333333年
国籍	cnt0000022   日本		
本籍地	prf000002   愛媛県		
	配偶者	<input type="radio"/> 有 <input checked="" type="radio"/> 無	
<input type="button" value="更新"/> <input type="button" value="クリア"/>			

<画面の作成例>



- 画面共通モジュールの詳細については、APIリストの「スクリプト開発モデル」-「画面共通モジュール」を参照してください。
- また、本書第3章「さまざまなコンポーネント群(im-BizAPI)の利用」の「18 エクステンション・モジュールの組み込みと操作」も参照してください。



### 3.16.1.2 グラフ描画モジュール(プレゼンテーション・ページ)



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。

```
<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。

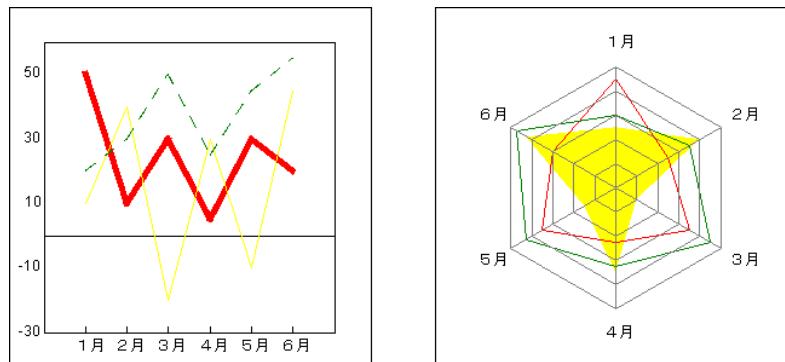
```
//バインド変数宣言
var oData = new Object(); // 折れ線グラフ描画データ

// 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>



### 3.16.1.3 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できます。

- ❖ 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。

アクセスコントローラ名	表示名
controller1	controller2
controller2	controller2

ロール	組織	役職	パブリックグループ
roleA			
roleB			



## 3.16.2 ビジネスロジック層

ビジネスロジック層に属するモジュールを説明します。



### 3.16.2.1 アプリケーション共通モジュール

各アプリケーション開発に必要な処理ロジックのモジュールが、使いやすくオブジェクト化されています。これらオブジェクトをビジネスロジックに組み込んで編集していくことで、余計なロジックの作り込みをせずに多階層アーキテクチャに基づいたWebシステムの開発を短期間でおこなうことが可能になります。

#### ■ 提供されるアプリケーション共通モジュールのオブジェクト概要

アプリケーション開発に必要な処理ロジックのモジュール（セッション管理やDBアクセス）が使いやすくオブジェクト化されて提供されています。このオブジェクトの利用により、ページをまたいだセッション管理を実現することができます。またフッター等に利用する会社名称や各種データベースへのログインユーザ名等の情報も、このオブジェクトから利用できます。これらのオブジェクトをアプリケーションロジックに組み込んで編集していくことで、余計なロジックの作りこみをせずに複雑なWebシステムの開発を短期間でおこなうことが可能になります。その他、アプリケーションの環境変数を含めた各設定値へのアクセス用オブジェクト、区分コードへのアクセス用オブジェクト、データベース関連の汎用オブジェクト、日付関連オブジェクト、デバック関連オブジェクト、URL管理オブジェクトなど多数用意されています。これらにより、現在接続している社員コードや、直前に表示したHTMLページ名などさまざまな情報にアクセスできます。

また、複数DBへの同時アクセスメソッドや大量の検索データに対して、指定した件数ごとに結果を画面表示する機能の検索ストリーミング、アプリケーション・ロック機能、XML対応モジュールなど、高度な機能が含まれています。

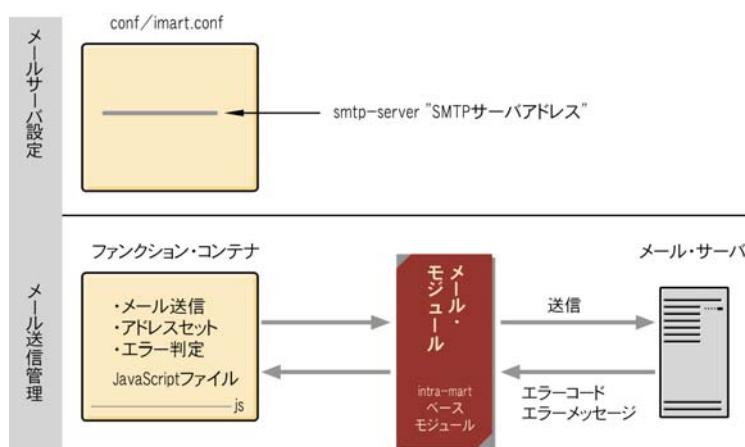


- アプリケーション共通モジュールの詳細については、APIリストの「スクリプト開発モデル」-「アプリケーション共通モジュール」を参照してください。



### 3.16.2.2 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。



## ■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("mail1000@nttdata.co.jp");
mailSender.addTo("mail1001@nttdata.co.jp");
mailSender.addTo("mail1002@nttdata.co.jp");
mailSender.addTo("mail1003@nttdata.co.jp");
mailSender.addTo("mail1004@nttdata.co.jp");

// CCメールアドレスをセット
mailSender.addCc("mail1005@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 );
}
```

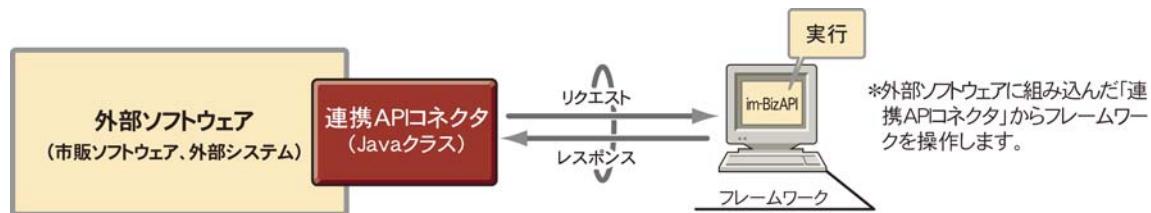


### 3.16.2.3 外部ソフトウェア接続モジュール

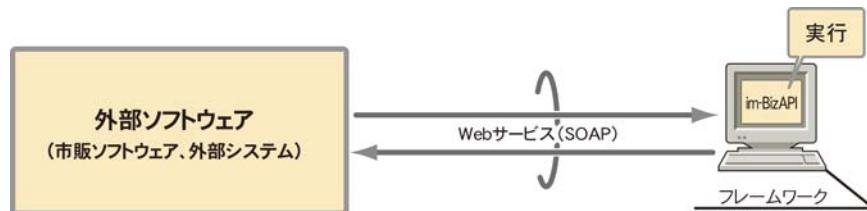


市販のアプリケーションパッケージからim-BizAPIの各種APIを呼び出して直接利用するなど、intra-martと外部ソフトウェアを簡単に連携・接続できるモジュールです。連携・接続する方法には、次の2通りの方法が用意されています。

ひとつは、この「連携APIコネクタ」がフレームワークの中のim-BizAPIと連携するためのJavaベースAPIとして提供されているので、外部ソフトウェアがJava実行環境であれば任意のプロセスとim-BizAPIを連携させる方法です。例えば、市販のポータルサーバ製品と組み合わせて、ポータル画面中にintra-martの画面を表示したり、他のアプリケーションと連携してバッチ動作する独自のJavaプロセスからユーザーアカウント情報を操作したりすることができます。



2つ目の方法としては、外部ソフトウェアからWebサービスによりim-BizAPIの各種APIを呼び出すことも可能です。



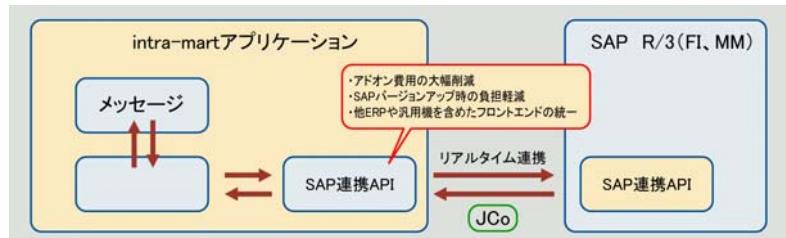
- このほかに、XMLによるデータ連携の方法もあります。詳細に関しましては、本書「8 XML形式のデータを扱う」を参照してください。
- 外部ソフトウェア接続モジュールの詳細に関しましては、APIリストの、「Developer's Guide」の「外部ソフトウェア接続モジュール」を参照してください。



### 3.16.2.4 ERP連携モジュール



SAP JCo技術を利用し、SAP APIをライブラリ化しました。標準のJava技術を用いて、アドオン開発を低コストで行えます。※intra-mart WebPlatform/AppFramework Enterprise版のみの機能です。



- 現在提供しているERP連携モジュールには、SAPとの連携ができる豊富なAPIが用意されています。その他のERP用の連携モジュールも順次追加して行く予定です。
- 詳細に関しては以下の資料を参照してください。
  - ・製品に添付されている、SAP R/3とリアルタイムに連携するためのチュートリアルガイド  
「ERP連携モジュール・チュートリアルガイド」(im\_sap\_api\_tutorial\_v61.pdf)
  - ・APIリストの「スクリプト開発モデル」「ビジネスロジック層」「ERP連携モジュール」
- SAP、SAP R/3、SAP JCO、製品内に記載するSAPの製品/サービス名は、すべてドイツおよびその他の国におけるSAP AGの商標または登録商標です。



### 3.16.3 業務基盤ツール



業務基盤ツールの属するモジュールを説明します。

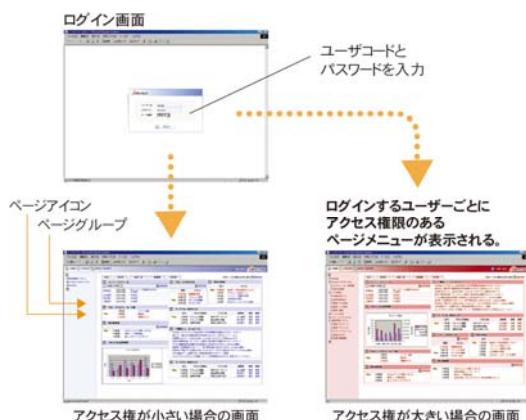


#### 3.16.3.1 アクセスセキュリティ・モジュール

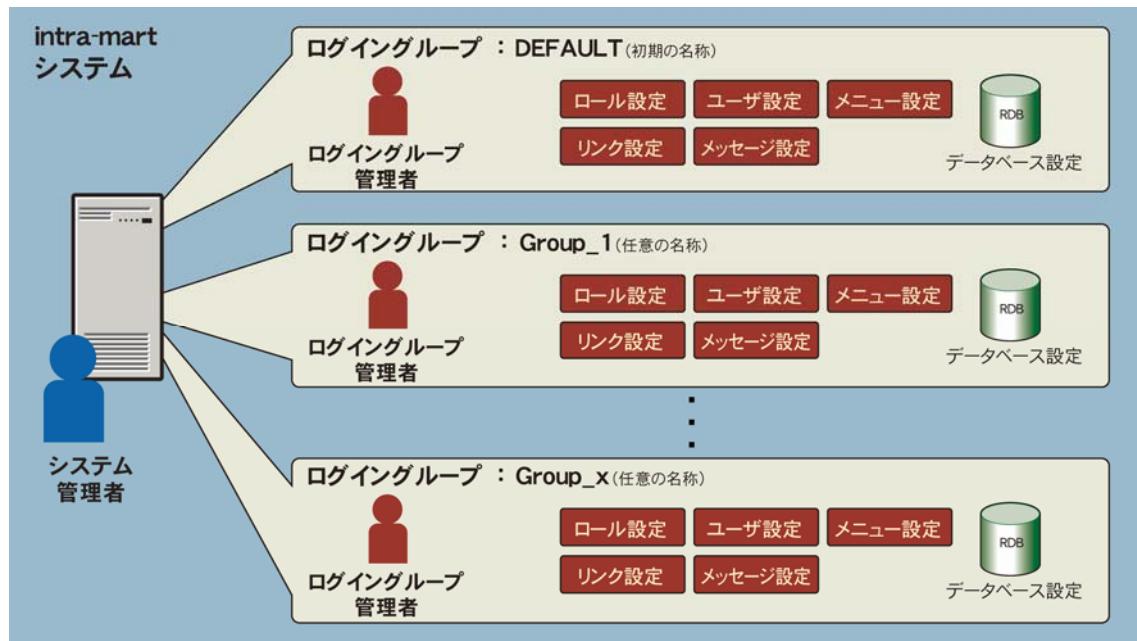


ユーザやロールなどのログイン・セキュリティ情報を操作するためのモジュールです。ログインするユーザの認証機能はもちろん、ユーザのアクセス権に応じた固有のWebページを表示することができます。たとえば、一般社員がアクセスしたときには、アクセス権限のないページはメニューにもあらわれないため、ユーザはその存在を一切意識することはありません。しかし、管理者がアクセスした場合には、同じページでも管理者が利用できるメニューまで表示されることになります。アクセスセキュリティ・モジュールを利用することで、このようなユーザに応じて内容の異なるページ構成を組むことが可能になります。設定方法についてはアドミニストレータガイド第2章「5 アクセスセキュリティの管理」を参照してください。

標準で用意されているアクセスセキュリティ・モジュールのほかに、シングルサインオンを実現するエクステンション・モジュール「IM-SecureSignOn（別売）」も用意されています。



intra-mart Ver5.0から、ユーザごとの言語の切り替えや画面のテーマカラーの選択が可能となりました。また、これまでユーザ情報の設定はシステム管理者がすべてを行う使用でしたが、システム管理者の配下にログイングループ管理者を設定し、ログイングループ管理者がユーザ情報の設定を行えるようになりました。



さらに、アクセスセキュリティの各機能はAPI化されており、それらを利用して、独自のメニュー画面を作成することもできます。また、intra-martでは完全なSecure Sockets Layer(SSL)サポートを提供します。これにより送信されるコンテンツに対して、暗号化セキュリティをかけることが可能になります。



- ここで紹介したintra-mart標準のアクセスセキュリティ・モジュールのほかに、エクステンション・モジュール(別売)として、「IM-SecureSignOn」を用意しています。「IM-SecureSignOn」は、独自のエージェント型リバースプロキ一方式により適用範囲が広く、導入・運用が容易なシングルサインオンを実現します。詳細は、本書第3章「さまざまなコンポーネント群(im-BizAPI)の利用」の「18 エクステンション・モジュールの組み込みと操作」を参照してください。
- アクセスセキュリティの詳細に関しては、別冊の「アクセスセキュリティ仕様書」も参照してください。



### 3.16.3.2 ワークフロー・モジュール



本モジュールを利用すると、Webブラウザベースのワークフローを効率良く構築することができます。作成したアプリケーションを「タスク」としてドキュメント・ワークフローに登録するだけでワークフローに対応したアプリケーションとして利用できます。ワークフロー機能の詳細は、別冊の「ワークフローガイド」を参照してください。



### 3.16.3.3 ビジネスプロセスワークフローモジュール



申請・承認が中心となるドキュメント・ワークフロー（標準のワークフロー・モジュールやIM-ワークフローデザイナーなど）とは異なり、ビジネスプロセスワークフローモジュールでは業務処理のプロセスをあらかじめ登録しておくことで、業務処理をそのまま自動化することができます。このため、事務処理の適正化を図ることができ、作業効率を大幅に向上させることができます。ビジネスプロセスワークフロー機能の詳細は、別冊の「ワークフローガイド」を参照してください。



### 3.16.3.4 バッチ管理モジュール



intra-mart は、Schedule Serviceによるプログラム実行のスケジューリング機能を提供しています。バッチ実行したいロジックを記述したバッチプログラムを作成し、バッチ設定画面にて起動日時を設定してください。

バッチ管理に関しては、アドミニストレータガイド第2章「11 バッチ管理の操作」を参照してください。

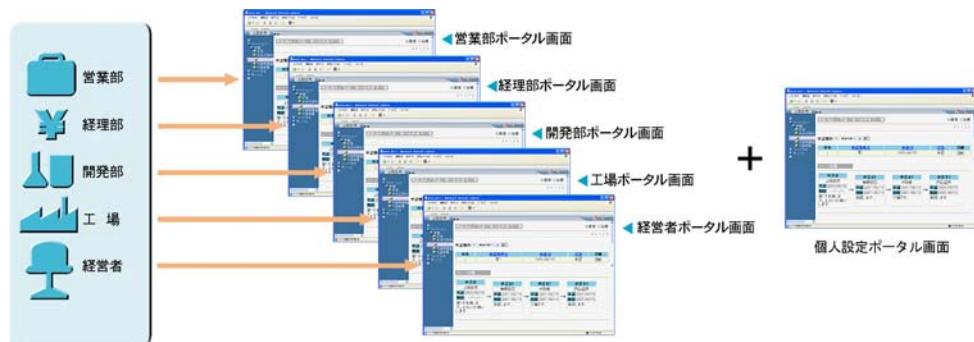


### 3.16.3.5 ポータルモジュール



ポータルモジュールは、intra-martのログイン初期画面にあらかじめ用意したページ（ポータルアプリケーションでは、ポートレットと呼びます）を表示させるモジュールです。よく利用するアプリケーションの画面やユーザに情報を見るように喚起する画面などを一覧表示することで、業務効率をあげることができます。

さまざまなアプリケーションで作成したページをポートレットとして自由にレイアウトして、ポータル画面を作成することができます。ポータル画面は複数作成することができ、ユーザが切り替えて表示することができます。さらに、組織やロール、ユーザによって利用できるポータル画面を切り替えることで、業務に最適な画面を提供することができます。

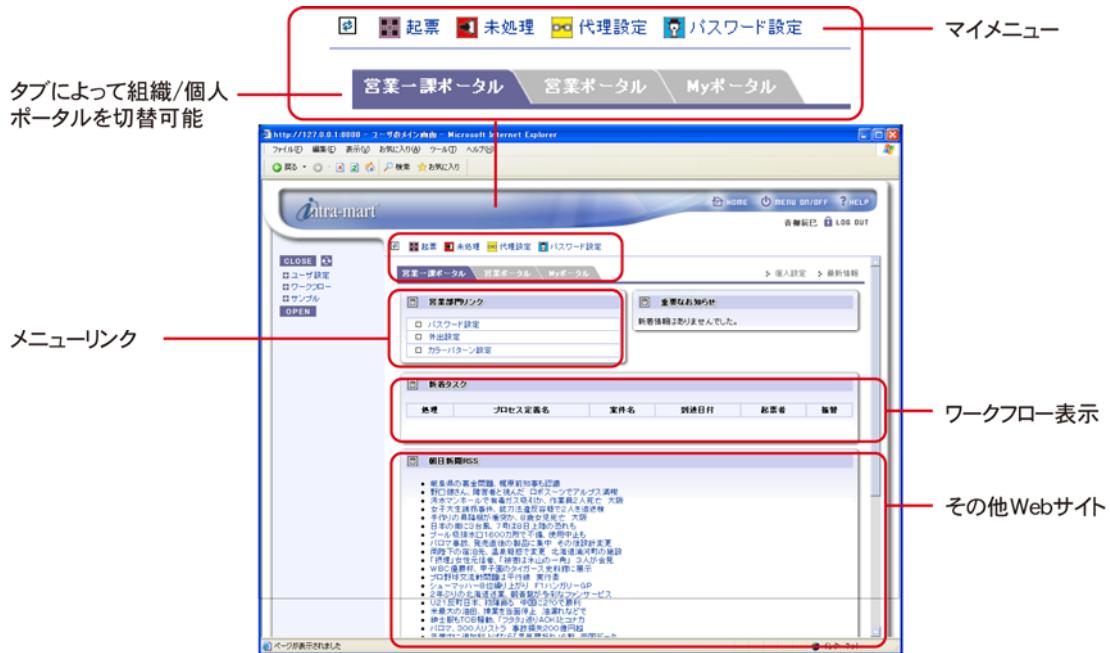


組織部署単位にポータルサイトを設定したり、個人単位の設定をすることで、組織内の情報伝達のスピードアップが図れる



会社(顧客・取引先)単位にポータルサイトを設定することにより、ECサイトやマーケットプレイスにおいて、サービスレベルを大きく向上させることができる

<会社・組織、ユーザごとのポータル>



## ■ ポートレット用ページの作成

スクリプト開発モデルおよびJavaEE開発モデル、外部URLでポートレット画面に表示させたいページをポートレット用に作成します。なお、intra-martには、あらかじめサンプルのポートレット用のページが用意されています。



### 3.16.3.6 ViewCreator

ViewCreatorはintra-martの画面上から、データベースのデータを使用して、様々な表やグラフを作成することができるツールです。使用可能なデータベースには、ログイングループデータベースとシステムデータベースがあります。ViewCreatorの機能は、クエリメンテナンスとデータ参照メンテナンスの2つの操作に大きく分かれます。

#### クエリメンテナンス

データベース上のテーブルやビューの結合などを行い、データ参照で使用する元データとなる表を作成します。また、作成したクエリのSQL表示やViewの作成、あるいは表をプレビュー表示することもできます。

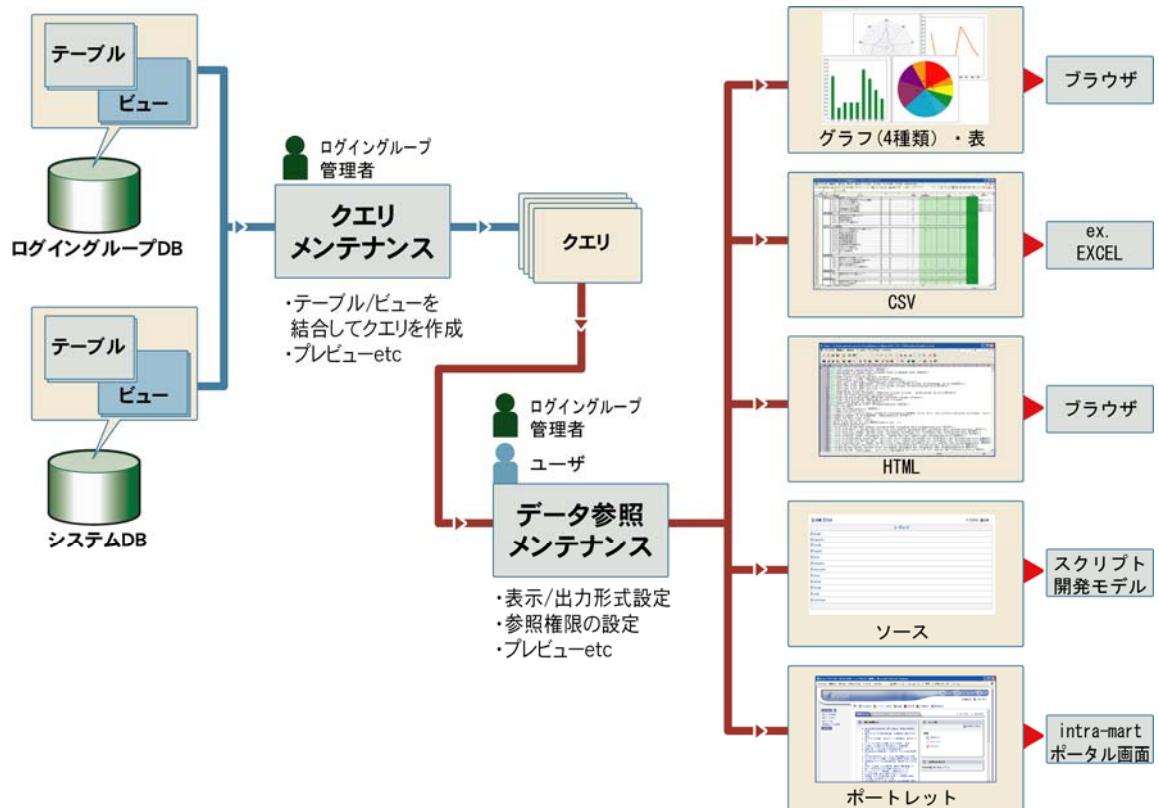
#### データ参照メンテナンス

クエリメンテナンスで作成したデータの表示の仕方(グラフまたは表など)やデータの絞込みに関する設定を行います。また、参照権限をデータ参照単位で設定することができます。

作成したデータ参照は、表示時にデータの検索や表示項目の絞り込み、並び順の変更などを行うことができます。また、データ参照はポートレットとして追加したり、CSV/HTML形式のファイルとして出力、あるいはスクリプト開発モデルのプログラムファイルとして出力可能で、出力されたプログラムソースは自由にカスタマイズして再利用が可能です。

このように、データベースの中のデータを元に様々な表やグラフの作成/表示をWebブラウザ上で簡単に操作できるのがViewCreatorの大きな特徴です。

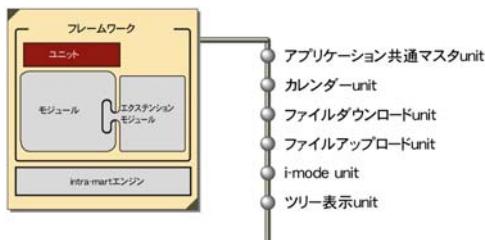
詳細は、アドミニストレータガイドの第2章「13 ViewCreator」を参照してください。



&lt;ViewCreator&gt;

# 3.17 ユニットの組み込みと操作

intra-mart WebPlatform/AppFrameworkには、再利用できるソフトウェア部品としてユニットが用意されています。これらは、以下のようにユーザーアプリケーションに組み込むだけで利用できるようになります。また、これらユニットはソースコードが公開されており、自由にカスタマイズすることができます。



## 3.17.1 アプリケーション共通マスタunit

会社データ、組織データ、グループデータ、取引先データ、顧客データ、商品データなど、システム開発でよく利用するマスタが標準で用意されています。これらのマスタを利用することにより、設計工程まで含め短期間でのシステム開発が可能となります。各intra-martアプリケーションシリーズと連携したシステムが開発でき、また、マスタにアクセスするためのAPIなども標準で用意されています。詳細については、アドミニストレータガイド第2章「8 アプリケーション共通マスタ」、および別冊「アプリケーション共通マスタ説明資料」を、アクセスするAPIは、APIリストの「アプリケーション共通マスタ API仕様」を参照してください。

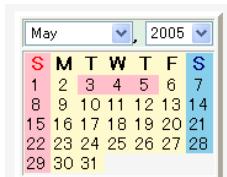


- intra-mart Ver5.0から、各マスタは日付で履歴管理することができます。



## 3.17.2 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>



### 3.17.2.1 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

```
<コーディング例>
<IMART type="link" page="@IM_CALENDAR_VIEW"
year="1999"
month="5">
</IMART>
```

上記コーディング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

<b>year (必須)</b>	カレンダーの表示年
<b>month (必須)</b>	カレンダーの表示月
<b>past (任意)</b>	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)
<b>future (任意)</b>	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future年間をコンボで選択可能)
<b>display (任意)</b>	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)
<b>link (任意)</b>	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、%ResourceService%/pages/src)からの相対パス)



### 3.17.2.2 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。

```
<コーディング例>
var sGroup = request.group; // カレンダーID
var sYear = request.dtyear; // 選択年の取得
var sMonth = request.dtmon; // 選択月の取得
var sDate = request.dtday; // 選択日の取得
```



### 3.17.2.3 カレンダー拡張タグとカレンダーモジュール

#### ■ カレンダー拡張<IMART>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。詳細は、「APIリスト」を参照してください。

#### ■ カレンダーモジュール

CalendarManager.\*

上記オブジェクトに含まれる各メソッドを利用してプログラムにおいてカレンダー設定情報を呼び出して利用することができるようになります。詳細は、「APIリスト」を参照してください。



- カレンダーマスタメンテナンスの操作に関しては、アドミニストレータガイドを参照してください。



### 3.17.3 ファイルダウンロードunit



ファイルダウンロードunitを組み込むと、サーバにあるファイルをWebブラウザを通してクライアントのパソコンにダウンロードすることができます。ダウンロードには、HTTPプロトコルを利用します。Storage Serviceを利用すると、ファイルを一元管理できます。



#### 3.17.3.1 ダウンロードの方法

ファンクション・コンテナでファイルダウンロードAPI (Module.download.\*) を利用してデータをクライアントへ送り出します。



#### 3.17.3.2 ファイル拡張子とMIMEタイプ

ダウンロード時には、ファイルの拡張子によって自動的にMIMEタイプが決定されます。

また、ダウンロードAPIへの引数の与え方により、ダウンロードAPIを利用するプログラム側で任意にMIMEタイプを指定することもできます。



- ファイルダウンロードunitは、ブラウザへデータを送信するための機能です。サーバ上のファイルをダウンロードするには、一度ファイルをロードしてから本unitを呼び出す必要があります。
- ダウンロードAPIは、クライアントへのデータの送信時に文字コード変換を行いません。クライアントが受信する文字コード形式への適切な変換は、ダウンロードAPIへデータを渡す前に、ダウンロードAPIを利用するプログラム側で行うようにしてください。
- ファイルダウンロードunitを利用したコーディング方法については、本書 第3章の「1 Storage Serviceの利用方法」を参照してください。



### 3.17.4 ファイルアップロードunit

ファイルアップロードunitを組み込むと、クライアントのパソコンにあるファイルをWebブラウザを通してサーバにアップロードすることができます。アップロードには、HTTPプロトコルを利用します。

Storage Serviceを利用すると、ファイルを一元管理できます。



### 3.17.4.1 プレゼンテーション・ページとの連携

プレゼンテーション・ページでは、以下のようにフォームを構築します。

```
<IMART type="form" method="POST" enctype="multipart/form-data">
  <INPUT type="file">
  <INPUT type="submit">
</IMART>
```



### 3.17.4.2 情報の取得方法

通常のリンクやフォームと同様に、ファンクション・コンテナ内においてrequestオブジェクトにてフォームの内容を取得できます。



- ファイルアップロードunitは、ブラウザからアップロードされたファイルの内容をデータとして取得するための機能です。アップロードされたファイルをサーバ上に保存するには、別途APIを利用する必要があります。
- ファイルアップロードunitを利用したコーディング方法については、本書 第3章の「1 Storage Serviceの利用方法」を参照してください。



### 3.17.5 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詳細については、APIリストの[スクリプト開発モール]-[ユニット]-[ツリー表示モジュール]を参照してください。



&lt;ツリー表示ユニットの例&gt;



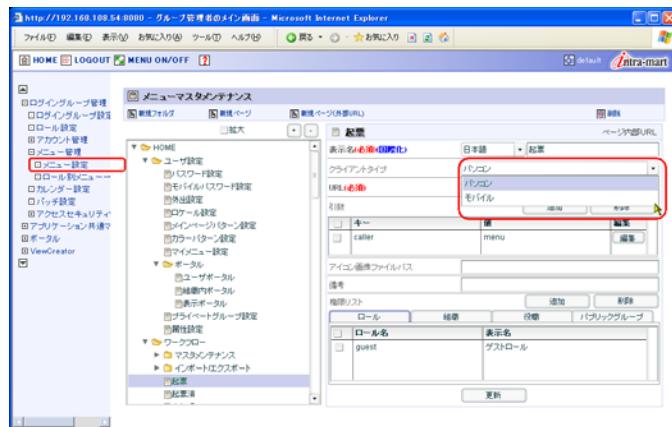
### 3.17.6 i-mode unit

i-mode unitを組み込むと、ユーザが作成し、ログイングループ管理者の[ログイングループ管理]-[メニュー管理]-[メニュー設定]で登録したページをi-mode対応の携帯電話の画面で見ることができるようになります。i-modeに対応した画面を作成するには、i-modeの表示領域に合ったページを作成し、[ログイングループ管理設定]-[メニュー管理]-[メニュー設定]で登録する際に、クライアントタイプで、「パソコン」または「モバイル」を選択します。また、利用するユーザの設定としては、「ユーザ設定」の「モバイルパスワード設定」で、あらかじめいくつかの項目を設定しておく必要があります。設定の詳細は、次項の「ページ管理マスタメンテナンスでのi-modeの設定」を、アクセスするAPIはAPIリストの「Module.mobile」を参照してください。



#### 3.17.6.1 ページ管理マスタメンテナンスでのi-modeの設定

作成したi-mode対応のページは、ログイングループ管理者が[ログイングループ管理]-[メニュー管理]-[メニュー設定]でページを登録する際に、「クライアントタイプ」の項目で「モバイル」を選択します。[メニュー設定]の操作については、アドミニストレータガイド第2章「7 アプリケーションの登録」を参照してください。



&lt;[ログイングループ管理]-[メニュー管理]-[メニュー設定]&gt;



### 3.17.6.2 i-modeのアドレスとパスワードの設定

i-mode用に作成し登録したページをi-mode対応モバイルで利用するには、ログイングループ管理者が[ログイングループ設定]-[アカウント管理]-[アカウント設定]で、ユーザごとにモバイル用メールアドレスとモバイル用パスワードの設定を行います。いったん設定されたモバイル用パスワードに関しては、各ユーザが一般ユーザでログインして、[ユーザ設定]の[モバイルパスワード設定]で変更することができます。

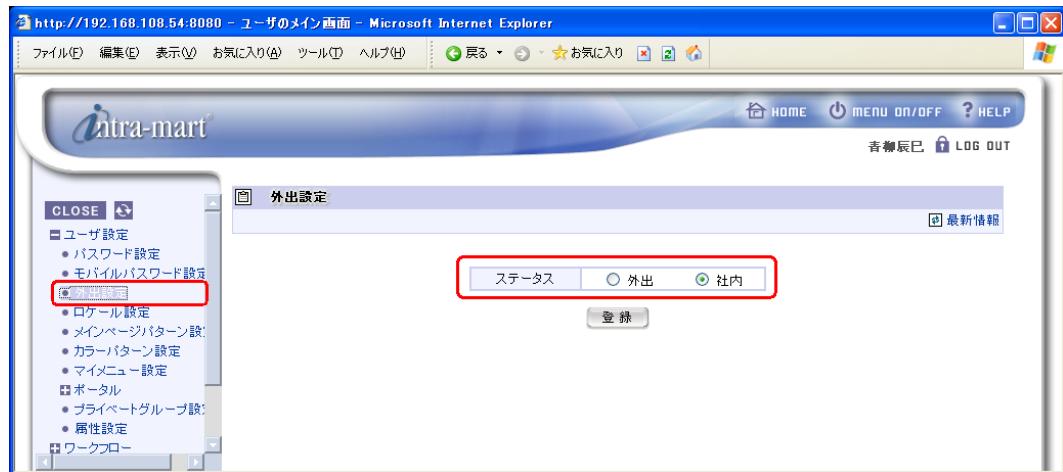
<ログイングループ管理者の[ログイングループ設定]-[アカウント管理]-[アカウント設定]>

<[ユーザ設定]-[モバイルパスワード設定]>



### 3.17.6.3 i-mode用外出設定

intra-martから送信されるメールは、通常時は[ログイングループ設定]-[アカウント管理]-[アカウント設定]で登録したメールアドレスに送信されます。一般ユーザが [ユーザ設定] - [外出設定] で「外出」に設定しておくと、通常のメールアドレスに送信されると同時に、i-mode用メールアドレスにも送信されます。この機能を利用するには、ログイングループ管理者が[ログイングループ設定]-[アカウント管理]-[アカウント設定]で [携帯メールアドレス] を設定しておかなければなりません。



&lt;外出設定&gt;



- i-modeで、警告画面を使用するときは、Module.mobile.alert()を使用してください。詳細は、「APIリスト」を参照してください。

エクステンション・モジュールは、intra-mart WebPlatform/AppFrameworkに標準添付されているモジュールとは別に用意されているモジュール群です。より高機能なモジュールが必要なユーザのために各種エクステンション・モジュールが用意されており、必要に応じて組み込んで標準のモジュールと同様に利用できます。現在用意されているモジュールについて説明します。



### 3.18.1 帳票印刷モジュール拡張

よりきめ細かな帳票の印刷に対応したエクステンション・モジュールがオプションで用意されています。



#### 3.18.1.1 IM-PDFデザイナー



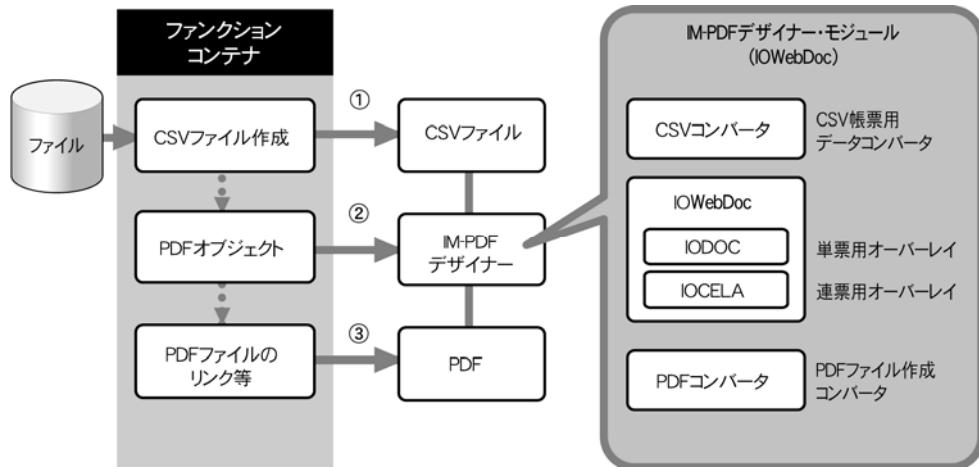
PDFを採用することにより、より複雑な帳票形式への対応が可能となるモジュールで、単票および連帳にも対応します。ビジュアルな帳票デザイン作成ツールである「IOWebDoc」で帳票フォーマットを作成します。ユーザアプリケーションからのデータはCSV形式で受け渡し、PDFファイルを作成し、Acrobatを起動して印刷します。

勤怠一覧表 1999年 5月分							1頁
							印刷日1999年5月6日
勤務地	NTTデータ	事業部	営業部	所属	営業第1課	業務	課長
社員番号	ueda	氏名	上田辰男	印			
年月日	休暇コード	業務コード1	業務コード2	始業／終業時間	実勤時間	休日時間	代休日
5 / 1 (土)	休日	移動		~			
5 / 2 (日)	休日	出張		~			
5 / 3 (月)	出勤日	客呼	客呼	9:30 ~ 18:30	8.00		
5 / 4 (火)	出勤日	設定	設定	9:30 ~ 18:30	8.00		
5 / 5 (水)	出勤日	バグ	バグ	9:30 ~ 18:30	8.00		
5 / 6 (木)	出勤日	通常	通常	9:30 ~ 18:30	8.00		
5 / 7 (金)	出勤日	会議	会議	9:30 ~ 18:30	8.00		
5 / 8 (土)	休日	教育	教育	~			
5 / 9 (日)	休日	設計	設計	~			
5 / 10 (月)	出勤日	実装	実装	9:30 ~ 18:30	8.00		
5 / 11 (火)	出勤日	移動	移動	9:30 ~ 18:30	8.00		
5 / 12 (水)	出勤日	出張	出張	9:30 ~ 18:30	8.00		
5 / 13 (木)	出勤日	客呼	客呼	9:30 ~ 18:30	8.00		
5 / 14 (金)	出勤日	設定	設定	9:30 ~ 18:30	8.00		
5 / 15 (土)	休日	バグ	バグ	~			
5 / 16 (日)	休日	通常	通常	~			
5 / 17 (月)	出勤日	会議	会議	9:30 ~ 18:30	8.00		
5 / 18 (火)	出勤日	教育	教育	9:30 ~ 18:30	8.00		
5 / 19 (水)	出勤日	設計	設計	9:30 ~ 18:30	8.00		
5 / 20 (木)	出勤日	実装	実装	9:30 ~ 18:30	8.00		
5 / 21 (金)	出勤日	移動	移動	9:30 ~ 18:30	8.00		
5 / 22 (土)	休日	出張	出張	~			
5 / 23 (日)	休日	客呼	客呼	~			
5 / 24 (月)	出勤日	設定	設定	9:30 ~ 18:30	8.00		
5 / 25 (火)	出勤日	バグ	バグ	9:30 ~ 18:30	8.00		
5 / 26 (水)	出勤日	通常	通常	9:30 ~ 18:30	8.00		
5 / 27 (木)	出勤日	会議	会議	9:30 ~ 18:30	8.00		
5 / 28 (金)	出勤日	教育	教育	9:30 ~ 18:30	8.00		
5 / 29 (土)	休日	設計	設計	~			
5 / 30 (日)	休日	実装	実装	~			
5 / 31 (月)	出勤日	移動	移動	9:30 ~ 18:30	8.00		
5月実勤時間		168 時間	5月実勤日数	21日			
年間実勤時間		168 時間	年間実勤日数	21日			

〈IM-PDFデザイナーを利用した帳票例〉

### ■ IM-PDFデザイナーの動作概要

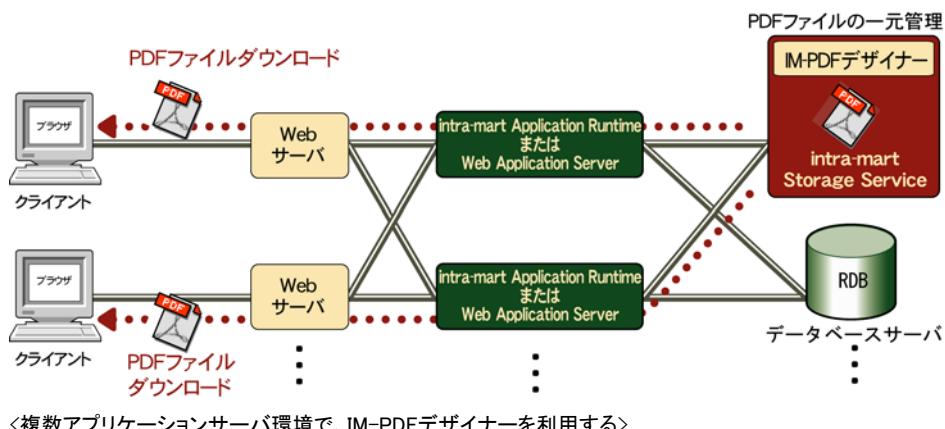
IM-PDFデザイナーを組み込むと、intra-martのモジュールの一つとして動作します。intra-martのアプリケーションのファンクション・コンテナから帳票データをCSVファイル形式で作成し、用意されているPDFオブジェクトを使用するだけで、より複雑な帳票をPDFで作成できます。



- 帳票のデザインはIM-PDFデザイナー(IMWebDoc)で作成します。
- IM-PDFデザイナーの制御ファイルおよびPDFオブジェクトの詳細については、「APIリスト」およびIM-PDFデザイナー付属のオンラインマニュアルを参照してください。

### ■ 複数アプリケーションサーバ環境でのIM-PDFデザイナーの利用

アプリケーションサーバを複数配置している環境でIM-PDFデザイナーを利用するには、Storage Service上でIM-PDFデザイナーを稼働します。これにより、生成したPDFファイルをStorage Serviceで一元管理することができます。





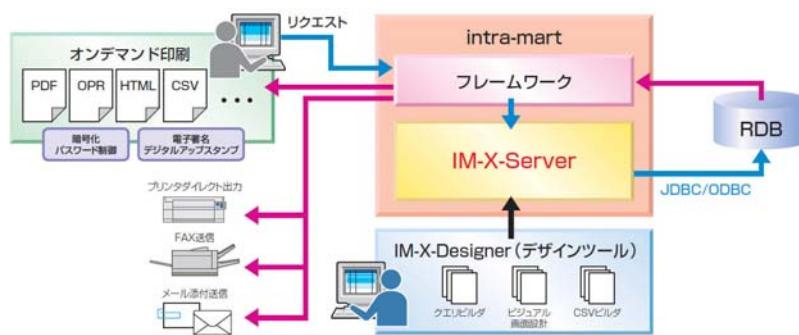
## 3.18.1.2 IM-X Server



大量帳票出力やプリンタへのダイレクト出力、電子署名とタイムスタンプのサポートも可能な印刷モジュールです。XML対応の高機能ビジネス帳票ソリューションで、オンデマンド印刷、ダイレクト出力など、多様な機能を提供します。

### ■ さまざまな形式の帳票の生成・出力・配信が可能

IM-X-Serverの帳票生成はXMLで定義されるため、電子化（PDF、HTML、CSV、OPR）や印刷（ダイレクト印刷、FAX送信）から検索・入力フォームまで、1ソースマルチフォーマットで生成でき、開発工数を削減することができます。基幹帳票生成から日々の業務に必要なビジネスレポートまで、トータルソリューションを提供します。



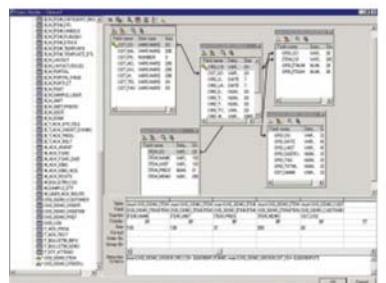
### ■ きめ細かい帳票作成が可能なデザインツールも合わせて提供

日本固有の複雑な罫線やページ単位のレポート、連帳・単票・サブレポート、ラベル、カスタムサイズ等の設計に対応。動的グラフ生成、動的バーコード生成、電子署名、表計算、複数オブジェクトのグループ化による段組表現など、フレキシブルな表現でさまざまなビジネス帳票に対応します。



[画面設計]

GUI部品の配置、プロパティ入力で直感的操作が可能。  
各種ウィザードの利用で設定不要



[クエリビルダ]

データベース種類を意識せずに設定が可能。  
GUIでDB、CSV、XMLのデータを取得。



[アクセス制御にも対応]

PDFサンプル/ドキュメントセキュリティ・電子署名対応。  
オンデマンドパスワード。



### 3.18.2 アクセスセキュリティ・モジュール拡張

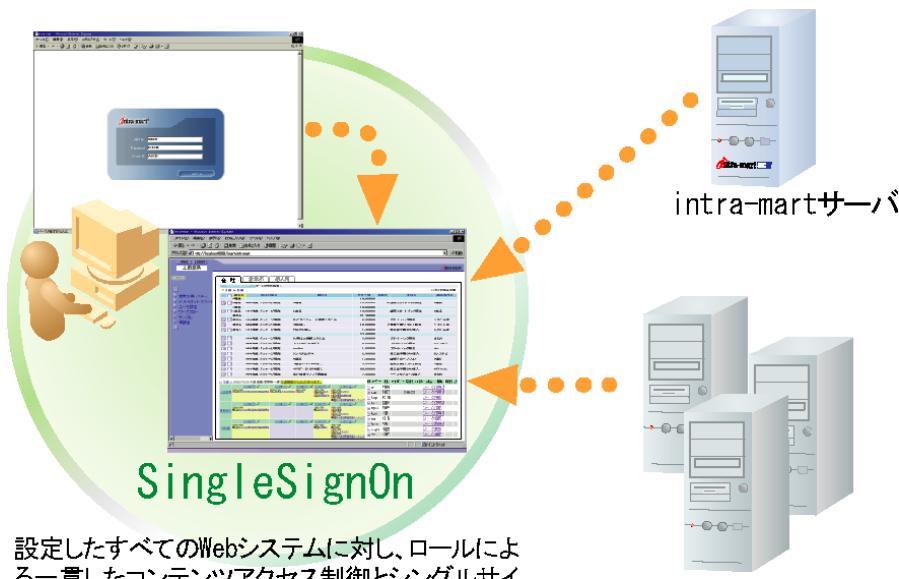
標準で用意されているアクセスセキュリティ・モジュールのほかに、シングルサインオンを実現するエクステンション・モジュールがオプションで用意されています。



#### 3.18.2.1 IM-SecureSignOn(セキュア・サイン・オン)



IM-SecureSignOnは、シングルサインオンを実現するツールです。社内のさまざまなWebシステムへのログインも、このSecureSignOnに一度ログインするだけ、すべての認証が完了します。独自のエージェント型リバースプロキシー方式により、適応範囲が広く、導入・運用が容易なシステムです。



#### ■ IM-SecureSignOnの特徴

- ❖ 大規模、異種環境下での利用が可能
- ❖ エージェント型リバースプロキシー方式により、「クライアントの設定が不要」「大規模環境に段階的な導入が可能」「Webサーバ、OSを問わない」というエージェントプラグイン、リバースプロキシー方式両者の特徴を備える。
- ❖ すでに多くのWebサーバが稼動しているときに段階的な導入が可能(すべてのアクセスを集中するProxyサーバは不要)
- ❖ ACLはサーバごとに分散管理している。(集中管理機能を構築中)
- ❖ ほぼWebサーバを選ばない方式である
- ❖ 認証モジュールはWindowsNT、Solaris、Linuxでの動作を確認済み。
- ❖ CGIを使って名前、メールアドレス、所属部署などのユーザ情報がとれる
- ❖ バックエンドユーザデータベースはLDAP及びNTドメインをサポート。プラグインにより独自データもサポート可能
- ❖ 電子署名によりアクセスチケットの改竄は不可能

## ■シングルサインオンを実現する3つの方式

シングルサインオンを実現するには、一般的にリバースプロキシー方式とエージェントモジュール方式の2つの方式が考えられます。IM-SecureSignOnでは、これら両者の長所を兼ね備えた独自のエージェント型リバースプロキシー方式を採用しています。

方式名	主な用途	長所	短所	システムの概念図
リバースプロキシー方式	インターネット、ASPなどに適したシングルサインオンの方式	<ul style="list-style-type: none"> <li>・セキュリティレベルが高い</li> <li>・エージェント不要</li> <li>・Webサーバに制約を受けない</li> <li>・ASPサービスに適用が容易</li> </ul>	<ul style="list-style-type: none"> <li>・認証サーバに負荷が集中</li> <li>・全てのアクセスが認証サーバを通る</li> <li>・Webブラウザの設定が必要</li> <li>・大規模ユーザに適さない</li> <li>・Webサーバの分散配置構成に適さない</li> </ul>	<pre> graph TD     A[HTTPサーバ] --&gt; B[SSO用プロキシー]     B --&gt; C[利用者]     D((アクセス権チェック))     </pre>
エージェントモジュール方式	管理集中型インターネットに適したシングルサインオンの方式	<ul style="list-style-type: none"> <li>・負荷が集中しない</li> <li>・各エージェントが直接アクセスを受ける</li> <li>・大規模ユーザに対応可能</li> <li>・複数ドメイン対応が容易</li> </ul>	<ul style="list-style-type: none"> <li>・Webサーバ一台一台に認証モジュールのインストールが必要</li> <li>・Webサーバに制約を受ける</li> </ul>	<pre> graph TD     A[HTTPサーバ] --&gt; B[認証プラグイン]     B --&gt; C[利用者]     D((アクセス権チェック))     </pre>
エージェントインストール型リバースプロキシー方式	管理分散型インターネットに適したシングルサインオンの方式	<ul style="list-style-type: none"> <li>・負荷が集中しない</li> <li>・各エージェントが直接アクセスを受ける</li> <li>・大規模ユーザに対応可能</li> <li>・Webサーバの分散配置構成に対応可能</li> <li>・Webサーバ、ホストOSの種類を問わない</li> </ul>	<ul style="list-style-type: none"> <li>・Webサーバ一台一台に認証モジュールのインストールが必要</li> </ul>	<pre> graph TD     A[HTTPサーバ] --&gt; B[認証モジュール]     B --&gt; C[利用者]     D((アクセス権チェック))     </pre>



- 詳細は、IM-SecureSignOn付属のマニュアルを参照してください。



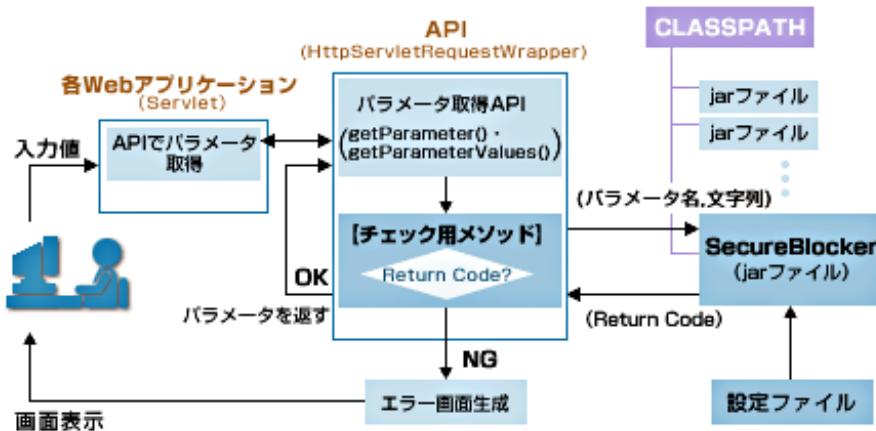
### 3.18.2.2 IM-SecureBlocker



インターネットに公開するWebアプリケーションのセキュリティ対策を低コストで実現することができます。脆弱な箇所にピンポイントで適用が可能なため、導入によるWebアプリケーション全体への影響を最小限に抑えることができます。

SecureBlockerは入力パラメータを検査し、自動的に無害化してくれるJavaクラスライブラリです。

API内部に実装するため、Servlet開発者からは隠蔽され、個別の修正が不要です。



IM-SecureBlockerを導入すると、次のような効果が得られます。

- ❖ HTTP Request のパラメータの入力値チェック機能の検討・実装コストが削減できます。
- ❖ 開発者のスキルに依存せず、Web アプリケーション脆弱性対策が実施できます。
- ❖ Web サイトで使用するパラメータごとにチェックを制御可能です。
- ❖ クロスサイトスクリプティング、OS コマンドインジェクション、ディレクトリトラバーサル、SQL インジェクションのチェックが可能です。



### 3.18.3 ワークフロー・モジュール拡張

標準で用意されているワークフロー・モジュールのほかに、エクステンション・モジュールとして「IM-ワークフローデザイナー（別売）」と「IM-FormatCreator（別売）」、「IM-Σ Serv／IM-SonicESB（別売り）」が用意されています。



- ビジネスプロセスワークフローはアドバンスド版に同梱です。



#### 3.18.3.1 IM-ワークフローデザイナー

プロセスが並列、結合、条件分岐するなど、より複雑な状況にも対応するワークフロー機能です。プロセス結合時や分岐する際の処理は、分岐処理、結合処理として設定し実行することができます。



- IM-ワークフローデザイナーの詳細は、別冊の「ワークフローガイド」を参照してください。



#### 3.18.3.2 IM-FormatCreator

intra-mart上で動作するドキュメント・ワークフローの起票画面（申請書）を、ノンプログラミングで作成するためのソリューション。ウィザード形式の画面にしたがって設定するだけで、ワークフローの申請画面を作成することができるので、HTML、JavaScript、XMLなどのWeb画面作成言語、およびデータベースに関する知識を必要としない。作成した申請書は、専用の連携画面で簡単にintra-martのワークフローと連携させることができます。また、以前に起票したものを利用することもできる。

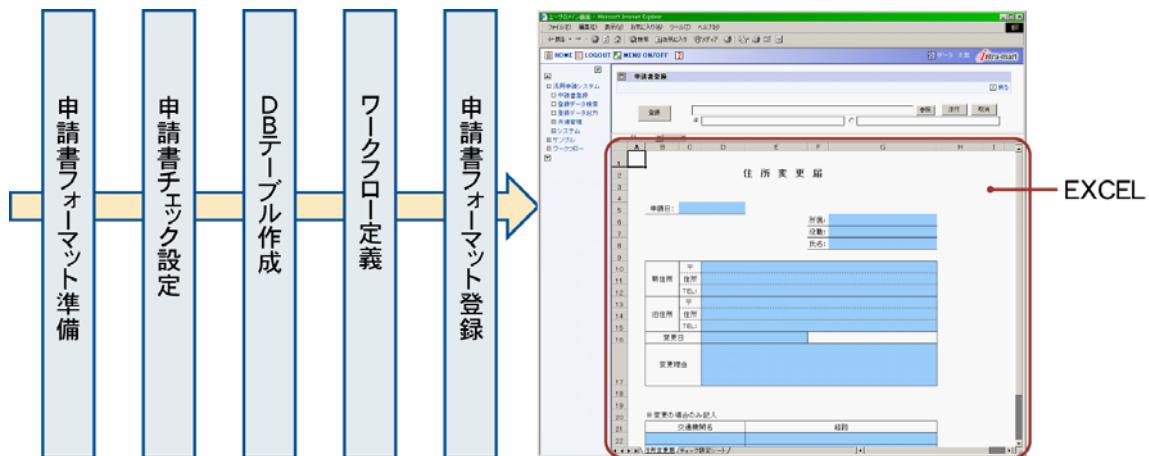


- 詳細は、別冊「IM-Format Creator」を参照してください。



#### 3.18.3.3 IM-EX申請システム

電子申請のための申請書フォーマット画面にExcelシートを利用することができます。intra-martのワークフローと組み合わせて利用することで、申請ワークフローを設定だけでノンプログラミングで作成することができます。書式設定やマクロはExcelの機能をそのまま利用することができるので、入力チェックを行うこともできます。入力した情報はデータベースの指定のテーブルに保存することができます。



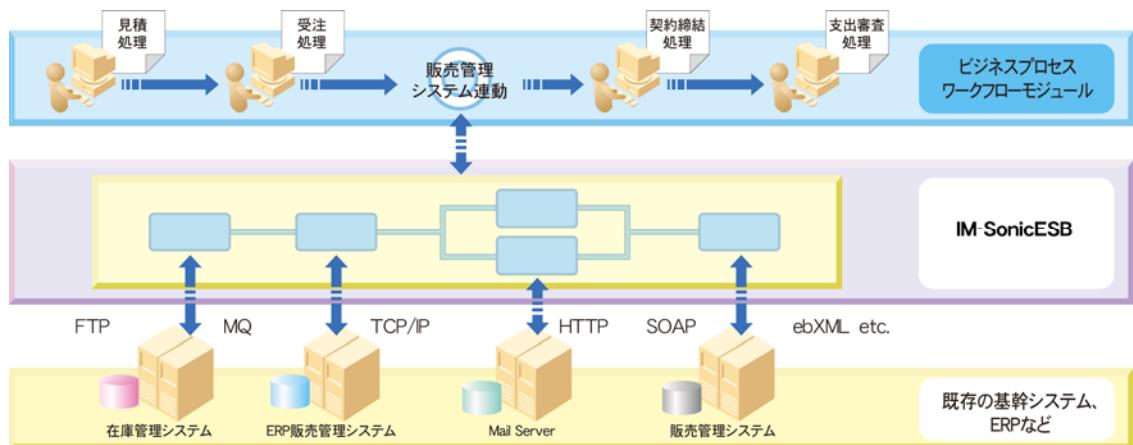
- ❖ Excel様式さえ用意すれば、短時間で申請書をWeb化することができます。
  - ❖ お使いのExcelの様式を入力画面に活用し intra-mart のワークフロー連動することができます。
  - ❖ Excelの書式設定や提供するマクロを設定し、クライアントサイドで入力チェックが行えます。
  - ❖ Excelに入力した情報は、データベースの指定のテーブルに保存可能です。
  - ❖ データベースに保存されている情報を指定のExcel様式で出力できます。
- Excel様式の登録、入力チェック等、すべて設定だけでプログラム言語の知識がいりません。



### 3.18.3.4 IM-SonicESB

IM-SonicESBは企業内に分散している複数の既存システムを連携する、バックエンドシステム統合プラットフォーム(ESB)です。システム間のデータの一貫性を保証する機能も実装しており、Webサービス同士を高信頼に連携させるアプリケーションを、容易に開発することができます。

ビジネスプロセスワークフローモジュールと連携することで、バックエンドも含めたダイナミックかつミッションクリティカルなシステム統合が実現します。





## 3.18.4 外部ソフトウェア連携ソリューション

各社が提供する外部ソフトウェアを利用するための連携ソリューションが用意されています。

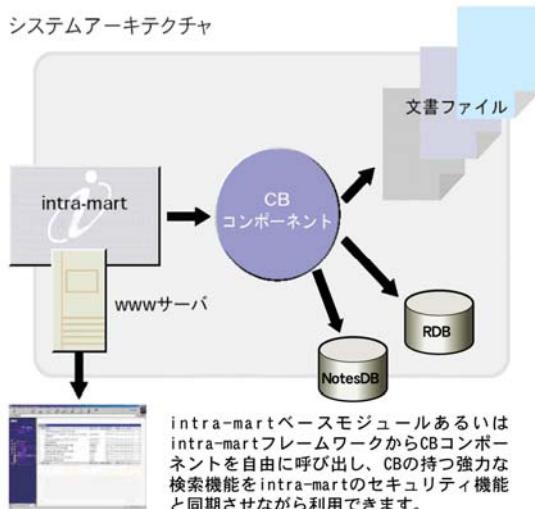


### 3.18.4.1 統合検索ソリューション

IM-統合検索ソリューションは、（株）ジャストシステムの「ConceptBase」と連携することで、MS-Word、一太郎、MS-Excel、PDFファイルなどさまざまなファイル形式のドキュメントから自然文によるナレッジ検索を行うことができる文書検索システムです。

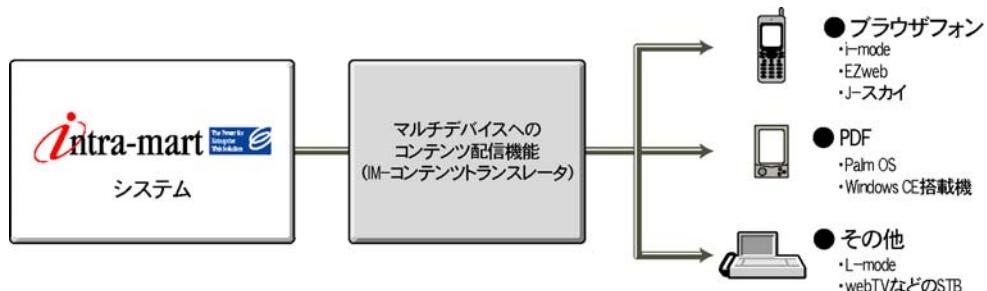
ConceptBaseサーバとStorage Service間の接続モジュールを提供することにより、ConceptBaseの強力な文書検索機能を利用したシステムの構築が可能となります。また、検索した文書から、内容やドキュメントの要約をテキストで取得することもできます。

Concept Base Searchは文書ファイルを対象とした検索、CB RDB GatewayはRDB内の情報を対象とした検索、CBゲートウェイfor Lotus NotesはNotes DBを対象とした検索が可能となります。



### 3.18.4.2 マルチデバイスソリューション

フレックスファーム社のX-Servletの利用により、マルチデバイスへのコンテンツ配信が可能になります。詳細については、X-Servlet付属のマニュアルを参照してください。



#### ■ 主な特徴

- |          |                                      |
|----------|--------------------------------------|
| シングルURL  | ひとつのURLにアクセスすると、自動的にアクセスされた機種を認識します。 |
| シングルソース  | 1つのコードですべての国内移動体通信会社のアクセス方式に対応します。   |
| オートレイアウト | アクセスされた機種に対応し、画像の変換やページの自動分割を行います。   |

intra-mart WebPlatform/AppFrameworkには、次のような機能も用意されています。



### 3.19.1 ライブラリ

intra-mart WebPlatform/AppFrameworkには、ユーザがソースコードに自由に手を入れてカスタマイズすることができる関数が用意されています。これらは、ファンクション・コンテナ内で [Module.メソッド名] または [Procedure.メソッド名] という形式で呼び出し可能です。詳細は、「APIリスト」の「アプリケーション共通モジュール」を参照してください。本機能は、スクリプト開発モデルのみが対象となります。



### 3.19.2 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。詳細については「APIリスト」を参照してください。

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



### 3.19.3 ソースセキュリティ機能

ソースセキュリティ機能とは、Webサーバ上には画像ファイル（gifなど）、CSJS、appletなどHTMLから直接参照しなければならない最低限度のファイルだけを配置し、メインのプログラムやデータはApplication RuntimeやResource Serviceの動作するService Platformに配置することで、Web経由での不正ファイルダウンロードを未然に防ぐための機能です。intra-mart は各サーバがネットワーク接続により別々のハードウェアで動作させることができるので、Webサーバと別マシンで動作しているApplication RuntimeやResource Serviceのプログラムファイル、またはStorage Serviceに保存されているデータファイルなどをWebを経由して不正にダウンロードすることができません。



### 3.19.4 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を利用して、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）。詳細は「APIリスト」を参照してください。



### 3.19.5 一意情報の取得機能

このAPIは、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る機能です（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）。詳細については「APIリスト」を参照してください。

```
Identifier.get()
```



### 3.19.6 データベースのストアドプロシージャの呼び出し

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。詳細は「API リスト」の「Developer's Guide」 - 「スクリプト開発モデル」 - 「ビジネス・ロジック層」 - 「アプリケーション共通モジュール」 - 「DatabaseManager」を参照してください。



### 3.19.7 ファイル操作

このアプリケーションは、Storage Serviceのシステムルート以下のファイルやディレクトリを操作するためのユーティリティです。ディレクトリやファイルの新規作成、ファイルの削除やアップロード、名称の変更、テキスト編集などが行えます。アップロードされたファイルはStorage Serviceに保存されます。

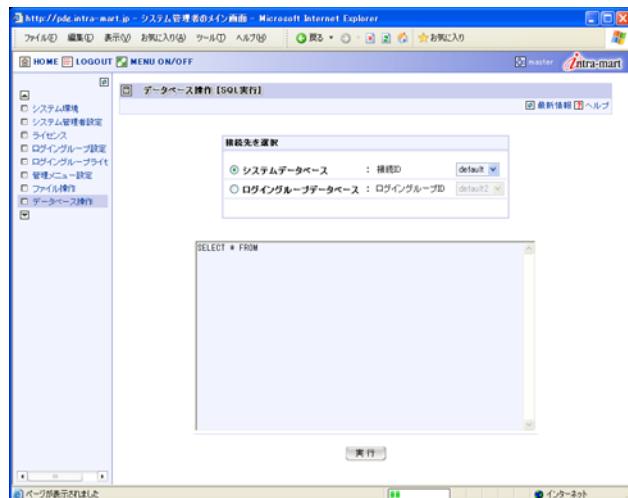
種別	ディレクトリ
ディレクトリ数	6
ファイル数	0
最終更新日	Mon May 23 11:24:50 GMT+0900 (JST) 2005

<ファイル操作画面>



### 3.19.8 データベース操作

データベース操作はデータベースに対してSQL文を直接実行するための簡易ツールです。接続先を選択後、テキストエリアにSQL文を記入し、[実行]ボタンをクリックします。



<データベース操作画面>



### 3.19.9 LDAPとの連携

LDAP上の情報に対し、検索、更新、削除などが行えます。詳しくはAPIリストの「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」－「Module.ldap」を参照してください。また、ログインユーザをintra-martとLDAP間で連携させることができます。アドミニストレータガイド第1章「11 LDAPとの連携」を参照してください。



### 3.19.10 國際化対応

以下の機能は、国際化 (i18n) を考慮した設計および実装がされています。

- ❖ スクリプト開発モデル
- ❖ im-JavaEE Framework(Service Framework のみ)
- ❖ アプリケーション共通マスタ
- ❖ アクセスセキュリティ・モジュール

これらを利用することにより、国際化されたアプリケーションの開発や、各アプリケーションの地域化などが実現できます。



- 利用するロケールの種類は、運用開始前に予め決定してください。
- 運用中にロケールを追加すると、特にアプリケーション共通マスタでデータの不整合が発生しアプリケーションの動作に影響する可能性があります。
- 機能の詳細については、下記ドキュメントをご覧ください。
  - ・APIリスト
  - ・im-JavaEE Framework仕様書
  - ・アプリケーション共通マスタ説明資料
  - ・アクセスセキュリティ仕様書



### 3.19.11 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。



### 3.19.12 ポータル画面の利用

intra-martのアプリケーションを作成する際に、独自のポートレットを作成して、intra-martのポータル画面に表示させることができます。

- ポータルとして表示させたい画面は、通常の画面ではなく、ポータルモジュールの規約にともない変更する必要があります。ポータル用のページ作成方法については、APIリストの[JavaEE開発モデル]-[業務基盤ツール]-[ポータルモジュール]-[ポータル]-[開発ガイド]および[スクリプト開発モデル]-[業務基盤ツール]-[ポータルモジュール]-[ポータル]-[開発ガイド]を参照してください。
- ポートレットに関しては、アドミニストレータガイドの第2章「ログインループ管理者編」の「10 ポータルの設定と操作」も参照してください。



### 3.19.13 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できます。

また、表示に関するセキュリティも設定することができます。

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>
```

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。

```
// 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";

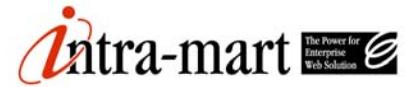
// 表示許可を行うユーザ
shortCutInfo.allowUsers = new Array("guest", "ueda");

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





**intra-mart WebPlatform/  
AppFramework**

## 第4章 Appendix

## 4.1

# 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、「アクセスセキュリティ仕様書」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。

## 4.2

# 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

# 4.3

## 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。



### 4.3.1 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。



### 4.3.2 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)
```



### 4.3.3 JS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。

## » 2

2つの Web アプリケーションモデル ..... 8

## » A

API リスト ..... 7

## » C

Client オブジェクト ..... 21

## » D

DatabaseManager オブジェクト ..... 24, 31

## » E

E4X ..... 63

EJB ..... 56

EJB コンポーネント ..... 56

ERP 連携モジュール ..... 100

## » F

FormatCreator ..... 118

## » H

Hello World ..... 14

## » I

IM- SecureBlocker ..... 117

IM- SecureSignOn ..... 115

IM- X Server ..... 114

IM-EX 申請システム ..... 118

im-JsUnit ..... 75

i-mode unit ..... 109

i-mode の設定 ..... 109

IM-PDF デザイナー ..... 112

IM-SonicESB ..... 119

IM-ビジネスプロセスワークフロー ..... 101

IM-ワークフローデザイナー ..... 118

IM-統合検索オプション ..... 120

## » J

JavaClass ..... 51

JavaEE 開発モデル ..... 8

JSSP-RPC ..... 67

## » L

LDAP との連携 ..... 123

## » V

ViewCreator ..... 103

## » X

XML パーサー ..... 58

XML 形式のデータ ..... 58

## » あ

アクセスコントローラモジュール ..... 96

アクセスセキュリティ・モジュール ..... 100

アプリケーション・ロック機能 ..... 121

アプリケーション開発概要 ..... 10

アプリケーション共通モジュール ..... 97

アプリケーション共通マスタ unit ..... 105

## » い

一意情報の取得機能 ..... 122

## » え

エクステンション・モジュール ..... 112

## » か

外出設定 ..... 111

外部ソフトウェア接続モジュール ..... 99

外部ソフトウェア連携ソリューション ..... 120

外部プロセス ..... 57

拡張<IMART>タグ ..... 47

画面共通モジュール ..... 93

カレンダーunit ..... 105

## » き

起動と終了 ..... 2

勤怠管理 ..... 87

勤怠修正 ..... 89

## » く

グラフ描画モジュール ..... 94

グローバル関数 ..... 49

## » け

検索ストリーミング機能 ..... 121

## » こ

国際化対応 ..... 123

## » さ

サンプル・アプリケーション ..... 86, 87

サンプルプログラムの実行 ..... 6

## » し

自動認証 ..... 3

ショートカットアクセス機能 ..... 124

## » す

スクリプト開発モデル	8
ストアドプロシージャ	122
ストレージサービス	33

## » セ

制限事項	130
セッションタイムアウト値	21

## » そ

ソースセキュリティ機能	121
-------------	-----

## » た

単体テスト	74
単体テストの実行	79

## » つ

ツリー表示 unit	109
------------	-----

## » て

データベースからデータを取得する	22
データベース操作	123
テストケースの作成	77
テストケースの実行順序	76
テ스트ランチャーの作成	79
デバッグ	71
デバッグ API	72
添付ファイル付きメール	44

## » は

バッチ管理モジュール	102
------------	-----

## » ひ

ビューエディタ	
ビューエディタの概要	103
標準画面	124

## » ふ

ファイル・アップロード	33
ファイル・ダウンロード	37
ファイルアップロード unit	107

ファイル操作	122
ファイルダウンロード unit	107
ファイルの削除	39
ファンクション・コンテナ	10
フォルダ	4
プレゼンテーション・ページ	10

## » ヘ

ページ	4
-----	---

## » ほ

ポータル画面	103, 124
ポータルモジュール	102
ホーム画面	5

## » ま

マルチデバイスソリューション	120
----------------	-----

## » め

メール送信	42
メール連携モジュール	97
メッセージ設定	128
メニュー構成	4
メニューの表示	4

## » も

モジュール	92
-------	----

## » ゆ

ユーザ定義関数	49
ユニット	105

## » よ

予約語	129
-----	-----

## » ら

ライブラリ	121
-------	-----

## » わ

ワークフロー・モジュール	101
--------------	-----



intra-mart WebPlatform/AppFramework

2009年9月 第四版

Ver6.1

プログラミングガイド  
スクリプト開発モデル編

株式会社 NTTデータ イントラマート  
〒107-0052 東京都港区2-17-22 赤坂ツインタワー本館  
TEL(03)5549-2821 FAX(03)5549-2816  
E-mail: info@intra-mart.jp  
ホームページ: <http://www.intra-mart.jp>

Copyright 2000-2007 株式会社NTTデータ イントラマート All rights Reserved.

※本マニュアルに記載されている社名および商品名は、一般に各社の商標および登録商標です。