Webアプリケーション概要

Webエンジン Ver. 3.0対応
リリース 1.1
初版:2003 年 3 月
改訂:2003 年 8 月

Muratec Information Systems.LTD.

Webアプリケーション概要、リリース 1.1
原本部品番号:W1A0001-02
原 本 名: Hayabusa Web Application Concepts、Release1
原本著者: 長谷川 和彦
編　　集: 久田 雅子

目次

はじめに

このマニュアルでは、Webアプリケーションの機能全般について説明します。このマニュアルによって、読者はWebアプリケーションの機能を理解することができます。

≪対象読者≫

このマニュアルは、Webエンジンのシステム管理者、アプリケーションの開発者を対象として記述しています。このマニュアルの読者は、システム管理、アプリケーション開発の概念に精通しているものと想定しています。
前提条件として、すべての読者は『Webアプリケーション概要』の第1章「Webアプリケーション概要」に記載されている内容を理解している必要があります。第1章では、このマニュアルのその他の章で使用される概念と用語について、幅広く説明されています。

≪本文の表記規則≫
本文中には、特別な用語が一目でわかるように様々な表記規則が使用されています。次の表は、本文の表記規則を示しています。

規則	意味
太字	太字は、本文中に定義されている用語または用語集に含まれている用語、あるいはその両方を示します。この句を指定する場合は、索引構成表を作成します。
大文字	大文字は、システムにより指定される要素を示します。
小文字	小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびサンプルのユーザー指定要素を示します。**注意:** 一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は、記載されているとおりに入力してください。
イタリック	イタリックは、プレースフォルダまたは変数を示します。

《コード例の表記規則》
次の表は、コード例の記載上の表記規則を示しています。

規則	意味
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。
|	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。
・・・ ・ ・ ・	省略記号は、例に直接関係のないコード部分が省略されていることを示します。

《アイコン》
本文中には、特別な情報を知らせるために、次のアイコンが用意されています。

ヒント
提案や秘訣を示し、これらによって、時間の節約や手順の容易化などを実現できる場合があります。

警告
システムに致命的な影響を及ぼす可能性のあるアクションについて、注意が必要であることを示します。

コラム
関連する基礎知識や細かい技などを解説しています。

第I部　Webアプリケーションの概要

『完成させるのが目的じゃないんですよね！
自然界に完成は無いんです・・・・・
いつまでも作り続けることが大切だと思うのです』

外尾悦郎サグラダファミリア主任彫刻家

ここでは、Webアプリケーションの概要について説明します。
構成は、次のとおりです。

第 1 章　Webアプリケーションの基礎知識
Webアプリケーションを理解する上で必要になる概念と機能について概説します。このマニュアルで説明されている詳細な情報を読む前に、この章をお読みください。

第1章 Webアプリケーションの基礎知識

この章では、Webアプリケーションを理解する上で必要になる概念と機能について概説します。このマニュアルで説明されている詳細な情報を読む前に、この章をお読みください。

1. Webアプリケーションの特長

近年、オリジナルの PDM パッケージを作成するにあたり、Web アプリケーションでの開発が不可欠です。その一方で、業務アプリケーションは、ますます複雑かつ短期集中型になりつつあります。そこで、あらかじめ標準的な機能をフレームワークとして提供しておき、個別に業務アプリケーションを構築していく事により、短期間に高品質なWebアプリケーションを開発することが可能になります。

①Webアプリケーションの共通機能

- 国際化対応
  国際化対応(ラベル、選択リスト、メッセージ)ラベル、選択リスト(HTML のメニュー)、メッセージなど、日本語、英語などの言語に依存するリソースを、アプリケーションからすべて分離し、プロパティーファイルで扱うようにしました。また、国際化機能は、フレームワークに組み込まれているため、アプリケーション開発者は、開発時にまったく意識する必要がなく、通常の SQL 文を投げるだけで、国際化対応の画面を作成する事ができます。

- 個人ポータルサイト作成
  ログイン時にユーザーを識別するため、個人ごとに自由に環境を設定することが可能です。個人宛のメールや、仕事残、スケジュール管理、進捗報告など、個人ごとに管理内容の異なるアプリケーションを簡単に導入する事ができます。

- アクセス制限の実施
  各画面に設けられた、ロール、グループ、その他の情報ごとに、リード／ライトを指定できます。ユーザーごとに画面に対して、アクセス制限をかけることが可能です。

- 電子メールによる簡易ワークフローの実現
  ユーザー情報として、ロール、グループ、プロジェクトを持っており、それらを利用してメールを送信することが可能です。メールの送受信には Java mail API を、また、オプションでメールサーバーとして、JAMES(Java Apache Mail Enterprise Server)を採用しました。

- DBユーザーと利用ユーザーとの分離
  Web アプリケーションでは、不特定多数のユーザーがログインする可能性があり、

また、HTTP のプロトコル上、ステートレスになっています（ステート管理は、クッキーや、JSP のセッションで管理）。本フレームワークでは、データベース接続に、コネクションプーリング技術を採用するにより、パフォーマンスを犠牲にせず、不特定多数の同時接続に対応しています。

②開発工数の削減

- データベースアクセス
  データベースに対して SQL を投げると、フレームワーク側で自動的に ResultSet から DBTableModel に変換し、JSP に渡して HTML 表示を行います。開発者が SQL 文をフレームワークに渡すだけで、その結果を HTML 画面に表示すると言う単純な方法です。複雑な業務ロジックの場合は、ストアドプロシージャ化する事により、業務ロジックの隠ぺいを行うことが可能になります。

- カスタマイズやシステム変更が容易な構造
  フレームワークを構築するにあたり、オブジェクト指向技術（Java の全面採用）を活用し、カスタマイズやシステム変更が容易な構造になっています。（但し、画面、データベース、業務ロジックなどのオブジェクト指向的でない過去からの資産が関係する部分は、オブジェクト指向的なアプローチは取っていません。）

- リソースファイル
  リソースファイル（初期設定）を利用して、開発工数を削減することができます。

③保守性の向上

- 画面（JSP）とデータ（ORACLE）と業務ロジック（SQL）の切分け／連携
  MVC（Model-View-Control）に明確に分ける事により、開発効率を高めました。画面の表現は、CSS（Cascading Style Sheets） を採用し、 HTML4.01 および、XHTML1.0 に準拠した Web ページを出力します。業務ロジックもView系はJSP、ﾊﾞｯﾁ系はPL／SQLと分けましたので、従来の開発手法、および、ノウハウは、そのまま利用可能です。

- データベースの作成
  データベース定義を先に作成し、そこからデータベース作成ファイルや、ロード定義、その他のツールを自動で作成することが可能です。また、運用中のデータベースの状態管理（エクステント、テーブルスペースなど）も、Web 上から簡単に管理する事ができます。

- ログレベルの設定
  JSP、JavaBeans共に、共通のログ出力機能を使用しています。また、ログ出力箇所ごとにログレベルを設定しておけば、外部からログレベルを指定することにより、簡易ログや、詳細ログを設定することが可能です。

- システム管理の遠隔（Webブラウザ）操作
  ユーザーのログイン制限や、接続プーリングのリフレッシュ、キャッシュデータの開

放など、一般的なシステム管理やシステム状態の監視を、遠隔から Web ブラウザ経由で行うことが可能です。

④販売戦略を意識

- 同時接続ユーザー数の制限(ライセンス販売時)
  本フレームワークのコネクションプーリング技術で、プール数をシステムプロパティーで設定できるようになっています。そのため、このフレームワーク上で同時接続数に制限をかけることにより、評価版と実稼動版の区別をつけたり、ライセンス販売時のコスト差に対応したパフォーマンスを提供させる事が可能です。

- BASIC認証とフォーム認証に対応
  ログイン時のセキュリティーは、インターネットでのWebアプリケーションのユーザー認証の基本である、BASIC認証とフォーム認証のどちらにも対応しています。個々の業務に合った方法で認証を行う事が可能です。

コラム　MVC(Model-View-Control) とフレームワーク

最近の Web アプリケーションでは、MVC を、Model=JavaBeans 、View=JSP、Control=Servlet と分けて考える傾向があります。もちろん、これは悪いことではなく、非常に『考え方としては』良いことです。
では、この考え方が、本当にベストかといわれると、色々な問題点が出てきます。
その中で2つの大きな問題点を挙げるとすると、1つは、アプリケーションが複雑になり、見通しが利きにくくなるというものです。もう1つは、Control とは、一体何か・・・ということです。
1つ目のアプリケーションの複雑化については、Web アプリケーションのコントロール自身が本当に複雑かと問われた場合は、私は、No と考えています。なぜなら、元々、リンク、ボタン、テキスト/メニュ入力 というプアーなインターフェースしか持たない Web アプリケーションにおいて、純粋な MVC に分けて管理する必要があるのか、それだけの価値があるのかという疑問です。まず、必要ないと考えています。
2つ目の問題点は、もっと抽象的です。(いわば、哲学の世界ですね。)
人が何か行おうとすると、一般には画面から操作を行います。また、画面から操作された命令を元に、アプリケーションがさらにコントロールします。しかし、どちらにしても、この操作のトリガ(きっかけ)は、画面であり、人による操作です。
このコントロールの結果、何らかの処理が行われて画面に結果が表示されます。
この、コントロールは、View の1機能ではないか・・・という考えが、この Web アプリケーションフレームワークにあります。つまり、コントロールの方法を変えたり、その結果、現れる画面も、どちらも View を修正する必要があり、さらに、Control を修正するというのは、管理対象が分散していることを示しています。管理は1箇所で行うべきで、で、あれば、変更が発生する箇所(つまり、View)で Control を管理すべきという結論になります。
もちろん、JSP で記述するカスタムタグ内部で Control するのであって、JSP に Control ロジックをゴリゴリ書くという趣旨ではないことを、付け加えておきます。

2. エンジンの全体構想

Web アプリケーションフレームワークのエンジン部分の構成を次に示します。ディレクトリ構成は、標準的なWebアプリケーションに準拠します。JSP/Servlet エンジンは、Tomcat を想定していますが、基本的にはどのエンジンでも動作するように標準的な作りになっています。

```
UAP(G:)     UAP(G:)ドライブ
├ bin/          各種実行アプリケーション、初期化ファイルディレクトリ
│  ├ init.bat          アプリケーション環境設定バッチファイル
│  ├ startup.bat       Tomcat  実行開始バッチファイル
│  └ shutdown.bat      Tomcat  実行終了バッチファイル
└ webapps/    web アプリケーションのルート(HOST ディレクトリ)
   └ dbdef/           DB 定義アプリケーションのドキュメントルート
      ├ filetemp/          ファイル UPLOAD 時のベース DIR
      ├ jsp/               JSP のトップ DIR(以下に業務ロジック A を配置)
      │  ├ common/              共通 JSP
      │  ├ custom/              アプリケーション固有共通 JSP
      │  ├ menu/                メニュー共通 JSP
      │  ├ TEST01/              画面 ID ごとの DIR
      │  ├ TEST02/
      │  ├ TEST03/
      │  │  ├ index.jsp         画面のトップ JSP(フレーム分割用)
      │  │  ├ query.jsp         検索入力画面(フレーム上段)
      │  │  └ result.jsp        結果表示画面(フレーム下段)
      │  │  ・・・
      │  ├ index.jsp     アプリケーショントップ JSP(フレーム3分割)
      │  └ index2.jsp    アプリケーショントップ JSP(折りたたみメニュー)
      │     ・・・
      ├ log/               実行ログ出力(Web アプリケーション分)
      ├ login/             フォーム認証ログイン用 JSP(web.xml で制御)
      ├ src/               ソースファイル
      │  └ resource/       国際化(リソース)管理関連
      │
      └ WEB-INF/ WEB-INF ディレクトリ
         ├ classes/        コンパイル済みクラスの配置
         │  └ resource/
         ├ lib/            ライブラリー(エンジン部分を最終的に配置する DIR)
         │  └ hayabusa.jar エンジン JAR クラスファイル
         ├ mis-taglib.tld カスタムタグライブラリ・ディスクリプタ
         └ web.xml         デプロイメント・ディスクリプタ


APPS(H:)    APPS(H:)ドライブ
└ java/          Java 関連アプリケーションのルート
   ├ jdk13/      jdk (JAVA_HOME)ルート
   │  └ jre/
   │     └ lib/
   │        └ ext/       各種拡張機能(外部アプリケーション)追加
   │
   └ tomcat/ Tomcat(CATALINA_HOME)ルート
      └ conf/           ファイル UPLOAD 時のベース DIR
         ├ server.xml  Tomcat 設定用のファイル
         └ web.xml     Tomcat 設定用のファイル
```

3. パッケージ

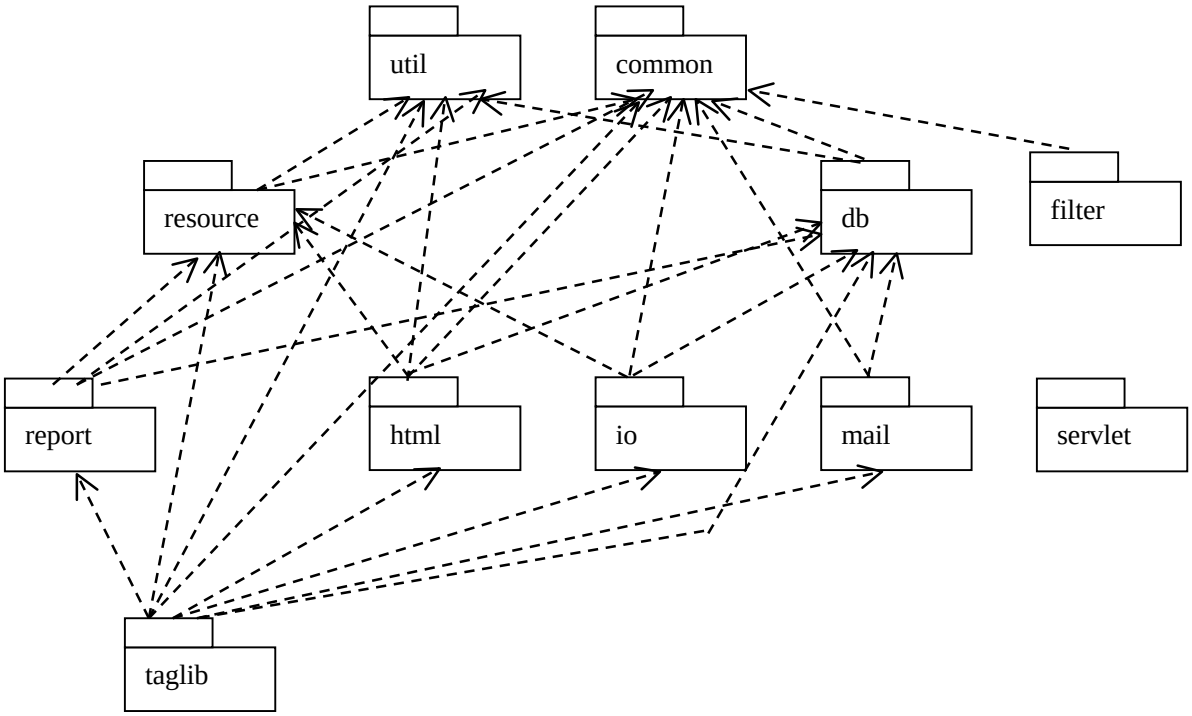
①パッケージ概要説明

Web アプリケーションフレームワークのエンジン部分のパッケージの概要説明を次に示します。

common	共通クラスフレームワークを利用する上でシステムリソースにアクセスしたり、例外（Exception）を加工したり、ログを制御したりする共通的に利用するクラス群です。このパッケージのクラスは、他のパッケージから独立しています。
db	DB アクセス関連、DBTableModel、 DBColumn 等データベースアクセス関連のクラス群です。特に、DBTableModel は、他のパッケージからも利用されるなど、フレームワークの中核をなすクラス（インターフェース）です。
filter	JSP のフィルターチェイン用のフィルタークラス群です。各種フィルターを用意しています。
html	HTML 関連ユーティリティ ViewForm インターフェースの具象クラス群と、XHTMLTag クラスを中心に、HTML 作成に特化した機能を提供するクラス群です。このクラス群を差し替えることで、XML 対応に簡単に移行できます。
io	ファイル入出力関連 DBTableModel をファイルに書き出したり、ファイルから読み込んだりするクラス群です。チェックイン/チェックアウトや EXCEL 出力などのコア機能を提供します。
mail	メール送受信関連メール関連のアクセス/ユーティリティクラス群です。現仕様は、メールデータも DBTableModel でやり取りしていますが、メール機能に特化したクラスに作り直す方向で考えています。独立したパッケージにして、Tagib からのみ使用する方向で検討中です。
report	帳票関連のクラス群です。印刷、PDF 化などを、EXCEL の雛型より作成できます。
resource	国際化（リソース）管理関連リソース関連のハンドリングを行うクラス群です。リソースそのものは、個々のアプリケーションごとに異なるため、mis.pdm.rk.resource 等の作番付パッケージで管理します。
servlet	Servlet 関連のユーティリティクラス群です。FileUpdate 時の処理など、Servlet 機能を利用するクラス群を分けています。 現時点では、純粋なサーブレットクラスは、使用していません。
taglib	画面表示用タグライブラリー画面の JSP からアクセスされるのは、この Taglib のみです。JSP のスクリプトは使用を誤るとメンテナンス性が落ちるため、必ず Taglib からアクセスさせます。よって、画面作成時に必要な機能が出た場合は、Taglib の開発をエンジンチームが行います。
util	共通ユーティリティークラスフレームワークと無関係に利用されるツール類（ユーティリティー）クラスです。本来、まったく別のパッケージにしても良いのですが、あえて利用するものだけを再度まとめなおしています。このパッケージのクラスは、他のパッケージから独立しています。

②パッケージ関連図

前に説明した Web アプリケーションフレームワークのエンジン部分のパッケージの関連図を次に示します。



4. 開発手順

フレームワークを利用して開発する場合の作業手順を次に示します。

①基本設計

基本設計は従来の開発と同様に行います。但し、次の資料は、従来、詳細設計のフェーズで作成していましたが、当フレームワークでは、基本設計の段階で作成する必要があります。

・DB 定義書（完成版・・・詳細設計で項目の変更可）
・画面一覧（画面 ID、名称、アクセス制限等の基本属性は必要）

DB 定義で使用するカラム名は、全テーブルにおいてユニークにする必要があります。さらに、カラム属性として、文字種別（S9、XL、XU、R など）と GUI 種別を定義しておく必要があります。また、フラグやコードなど指定値以外の登録ができないものについては、あらかじめすべての項目の定義値を登録しておく必要があります（プルダウンメニュー）。これらの情報は、リソースファイルとしてフレームワークで利

用されるので、開発に入る前に準備しておく必要があります。

②画面サンプル（納入仕様書）

当フレームワークでは、検索系画面作成については、簡単に直接 JSP で記述することができます（EXCEL や HTML で作成しなくてよい）。そのため、作業効率を大幅に向上させることができます。もちろん、すべての画面やすべての DB 項目が埋まる必要はなく、ユーザーと打合せながら、画面サンプルを作成していきます。詳細設計に入る前に、実際に動作する画面をすべて作成してしまうため、従来のウォーターフォール方式からスパイラル方式へと発想を切り替える必要があります。このフェーズで画面 ID 一覧、画面遷移図、DB 項目定義を完成させます。

③詳細設計

検索系については、前フェーズでほとんど開発は終了している状態なので、あえて仕様書を作成する必要はありません。登録系についても、DB 定義で各カラムの論理属性を定義しておくので、ほとんどの設計については、仕様書を作成する必要はありません。ここで従来と同様の仕様書が必要となるのは、業務ロジック B といわれる PL/SQL でのリアルタイムバッチ処理に関する仕様書になります。これは、JSP 画面から PL/SQL を CALL し、実行結果を直接/間接的に返して画面に表示しますが、PL/SQL の処理は一般に仕様展開や逆展開、差分などの高度な処理のため仕様書が必要となります。なお、ここでの仕様書は、開発のための仕様書であり、維持（保守）のための仕様書ではないため、このフェーズできっちり作成する必要はありません。また、フレームワークの機能不足により画面が作成できないケースが発生すると思われますが、この段階で、フレームワークチームと共同で、フレームワークの機能強化を行います。各プロジェクトで個別に対応する事を避けるとともに、専門チームによる品質の高い部品を標準機能として拡張していくことにより、次回以降の開発期間の短縮も図ります。

④開発／単体テスト

環境作成は、DB 定義や画面一覧を、フレームワーク上のテーブルに登録しておきますが、この作業は、すでに基本設計、または画面サンプル作成時点で終了しています。このデータを元に、予想データ件数を登録することにより DB 作成コマンドや容量計算を自動で行い、テーブル作成も自動化します。本番サーバーに対してテーブルを作成し、実データを開発前に導入すれば、すでに検索系は本番と同等の機能として実現できているため、データの不具合や画面項目不足、検索速度などの確認がすぐにできるようになります。そして、ユーザーの理解のもと、早い段階で動作を見てもらい、不具合点や機能不足の指摘を受けられるようにしておきます（その指摘に対して対応するかどうかは、機能の大きさと優先順位により決定）。実開発は、各画面のチューニング（使い勝手の向上やデザイン変更など）と、PL/SQL の実装になります。もちろん、JSP と PL/SQL との連結動作も確認しておく必要があります。

⑤総合テスト

総合テストは、従来の方法と同じく、全体機能を見ながら各モジュール間の連結不具合をチェックしていきます。これより前に、特定のユーザーにはオペレーションして頂いていますが、このフェーズでさらに、オペレーションのトレーニングを行う必要があります（操作マニュアルも必要）。ただし、これらは、もっと前に行うこともできます。

⑥本番導入/導入後サポート/維持仕様書作成

本番導入後安定するまで、導入後サポートを行います。また、この期間に、完成されたアプリケーションに関する正確な設計仕様書を作成します。ここでは、維持(保守)に必要な情報を仕様書にまとめていきます。最終ドキュメントは、ソースコードであり、そこにたどりつくための資料という位置付けなので、主に青紙の表紙（概要とDB アクセスと処理内容）と、画面遷移を作成します。

5. 使用ツール

フレームワークを構築するにあたり、各種ツール類を利用しています。以下に、各使用ツールと、概要説明を示します。

①本番ツール

本番ツールというカテゴリは、本 Web アプリケーション稼動における必須ツールです。必ずインストール/セッティングしておく必要があります。

- JDBC

・classes12.jar
JDBC は、Java からデータベースへアクセスするための API で、Type1～4 までの種類が存在します。これは、Type4ドライバ（Thinドライバ）です。
http://otn.oracle.co.jp/software/db_connect/jdbc/jdbc.html

・ojdbc14.jar
JDK1.4 用の ORACLE9.2 対応 Thin ドライバです。
OCI ドライバ（Type2）を利用する場合は、クライアントにインストールする DLL とこれらのドライバの整合性をあわせる必要がありますが、そうでない場合は、このドライバをしようする事が可能です。

・nls_charset12.jar
JDK 1.2.x で使用する、オブジェクト型およびコレクション型の NLS サポートに必要なクラスです。
http://otn.oracle.co.jp/software/db_connect/jdbc/81700/jdbc81700.html

- XML

・jaxp.jar (Java API for XML Parsing)
Java 言語用の標準 XML API です。
http://java.sun.com/xml/jaxp.html

・xalan.jar (ザラン)
Apache XML Project の一部として開発されている XSLT プロセッサです。
http://xml.apache.org/xalan-j/index.html

・xerces.jar (ザーシーズ)
Apache XML Project の一部として開発されている XML パーサです。
http://xml.apache.org/xerces2-j/index.html

・crimson.jar
crimson は、xerces よりもコンパクトな、XML パーサです。
http://java.sun.com/xml/

・xsu12.jar(XML SQL Utility)
XSU を使用すれば SQL 問合わせの結果から XML を生成することができます。
http://otn.oracle.co.jp/software/tech/xdk/xdk_java/8171a/xdk_java8171a.html

ヒント
JDK1.4 を使用する場合は，標準 API 化されていますので、
追加モジュールの導入は必要ありません。

- JavaMail

・mail.jar
インターネットメールを中心に、電子メール全般を扱う(送受信等)ために用意されたAPI仕様と、そのリファレンス実装のことです。
http://java.sun.com/products/javamail/
http://java.sun.com/j2ee/ja/javamail/index.html

・activation.jar (JavaBeans Activation Framework)
データの種類 (MIME タイプ)に応じて適切な JavaBeans を呼び出す仕組みを提供する Java API です。 Java Mail で使用します。
http://java.sun.com/products/javabeans/glasgow/jaf.html

ヒント
メール機能を使用しない場合には、インストール不要です。
ただし、標準機能として，いつでも使用できるようにインストールしておくことを、お薦めします。

②開発環境ツール

開発環境ツールは、本番実行とは無関係の機能ですので、インストール不要です。
ただし、今後、開発環境も含めて Web アプリケーション総合開発環境パッケージとして整理していきます。

・log4j.jar
バイナリのアプリケーションを修正することなく、実行時のログを記録することを可能にします。
http://jakarta.apache.org/log4j/docs/index.html
http://www.ingrid.org/jajakarta/log4j/

・junit.jar
テスト実行支援ツール
http://www.junit.org/

・JMeter
JMeter とは、テスト機能の動きに負荷をかけ、パフォーマンスを計測するためにデザインされた、 100%ピュア Java のデスクトップアプリケーションです。
http://jakarta.apache.org/jmeter/index.html
http://www.ingrid.org/jajakarta/jmeter/

・ant
Ant は、Java ベースのビルドツールです。理論的には、make の欠点がない make の一種です。
http://jakarta.apache.org/ant/index.html
http://www.ingrid.org/jajakarta/ant/

③本番使用拡張ツール（オプション）

必要時にインストールするオプションです。

・PDF995
プリンタドライバとして、出力ファイルを PDF ファイルに変換します。
新帳票システム等の PDF 帳票が必要なときに、サーバーにインストールする必要があります。
http://www.pdf995.com

・CODE39 バーコードフォント
帳票関連出力で、バーコードを出力する場合に、サーバー環境にバーコードフォントをインストールする必要があります。
http://www.technical.or.jp/handbook/index.html
http://www.technical.or.jp/usbbarscan/CODE39.ttf

④本番使用サーバー

本番サーバーは、JSP/Servlet を実行するために必須です。

順番としては最も大切なのですが、互換性が高ければ、どのサーバーを利用しても、Web アプリケーション実行環境として利用できます。

・tomcat4.0
Tomcat は、Java Servlet 2.3 および JavaServer Page 1.2 技術の正式なリファレンス実装です。
http://jakarta.apache.org/tomcat/index.html
http://www.ingrid.org/jajakarta/tomcat/

⑤テスト/評価サーバー

テスト/評価サーバーとは、本番環境にインストールして使用してもよく、かつ、本番環境と同等の環境が無い場合に，利用できるサーバー群です。

・james (JAVA APACHE MAIL ENTERPRISE SERVER)
Javaで実装された Mailサーバーです。(POP、SMTP)
http://jakarta.apache.org/james/index.html

・Apache HTTP
Server 最も普及している、HTTPサーバーです。
Tomcat と組み合わせることで、Web サーバー機能と JSP 実行機能を分離させて、パフォーマンス、セキュリティ、スケーラビリティーを確保することができます。
http://httpd.apache.org/

6. インストール手順

Webアプリケーションをインストールする手順を以下に示します。

1. インストール前に決定しておく必要のある情報

インストール前に下記の事前情報を調査、決定しておいてください。

・インストール先のOS	例) Windows 2000
・APPS ドライブ名	例) H:
・UAP ドライブ名	例) G:
・最大使用メモリ	例) -Xmx128m
・初期使用メモリ	例) -Xms64m
・認証用接続DB情報	例) jdbc:oracle:thin:@HN5191:1521:ORCL
・接続先DB情報	例) jdbc:oracle:thin:@hn5132:1521:HN5132

APPSドライブは、アプリケーションのインストールドライブです。
UAPドライブは、ユーザーアプリケーションドライブです。
通常 MISでは、APPS(H:) UAP(G:)となります。

2. アプリケーションのインストール(APPS)

CD－ROM より、APPSドライブに Java , Tomcat 等のフォルダをコピーします。

3. ユーザーアプリケーションのインストール(UAP)

CD－ROM より、UAPドライブに bin , webapps 等のフォルダをコピーします。

4. bin¥init.bat の書き換え（ドライブ設定、メモリ設定）

$UAP¥bin¥init.bat を書き換える。

```
set APPS=H:          ←  APPSのドライブ名
set UAP=G:           ←  UAPのドライブ名

set CATALINA_OPTS=-server -Xms64m -Xmx128m
      ↑   Java 起動時使用メモリ
     Xmsは、初期使用メモリ
     Xmxは、最大使用メモリ

        使用メモリは、Web アプリケーションのみの場合は、
                最大使用メモリ   実メモリの半分
                初期使用メモリ   最大使用メモリの半分
        を目安に、設定すれば問題ありません。
```

5. java¥tomcat¥conf¥server.xml の書き換え（ドライブ設定、認証DB設定）

$APPS¥java¥tomcat¥conf¥server.xml を書き換える。

PORT設定、DNS逆引き停止

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
        port="8080" minProcessors="5" maxProcessors="75"
        enableLookups="false" redirectPort="8443"
        acceptCount="10" debug="0" connectionTimeout="60000"/>
```

・ドライブ設定

```
<Host name="localhost" debug="0"
        appBase="G:¥webapps" unpackWARs="true">
                ↑  UAPのドライブ
```

・認証DB設定

```
<Realm   className="org.apache.catalina.realm.JDBCRealm" debug="99"
                driverName="oracle.jdbc.driver.OracleDriver"
                connectionName="GE"
                connectionPassword="GE"
                connectionURL="jdbc:oracle:thin:@HN51D4:1521:ORCL"
                userTable="GE10"
                userNameCol="userid" userCredCol="passwd"
                userRoleTable=" GE10" roleNameCol="role" />
```

・コンテキスト設定

```
<Context path="/dbdef2" docBase="dbdef2" debug="0"
                reloadable="true" crossContext="true">
```

```
<Logger className="org.apache.catalina.logger.FileLogger"
        prefix="localhost_dbdef2_log" suffix=".txt"
        timestamp="true"/>
```

6. SystemResource.properties の書き換え(接続DB設定)

本番環境のデータベースサーバーの設定を行います。
$UAP¥webapps¥プロジェクトID¥src¥resource¥SystemResource.properties
の、DEFAULT_DB_URL = jdbc:oracle:thin:@DBサーバー:1521:SID
を、変えてください。

例:DEFAULT_DB_URL = jdbc:oracle:thin:@hn5132:1521:HN5132

7. リソースファイルのコンパイル

$UAP¥webapps¥プロジェクトID¥src¥jccall.bat
を、ダブルクリックして、リソースファイルのコンパイルを行ってください。
なお、画面上からリソースファイルをコンパイルしている場合は、
起動用 jccall.bat は、絶対パスで記述する必要があります。
必要な場合は、各自書き換えて、テストしておいてください。

8. bin¥startup.bat のスタートアップメニューへの登録

TOMCAT は、サービス化していません。(未検証のため)
OS再起動時に自動的にWebサーバーを起動するために、スタートメニューに
startup.bat のショートカットを登録しておいてください。
Windows 2000 の場合は、『C:¥Documents and Settings¥All Users¥スタート メニュー¥プログラム¥スタートアップ』です。

9. Tomcat 実行

$UAP¥bin¥startup.bat をダブルクリックすれば、TOMCAT が起動します。
$UAP¥bin¥shutdown.bat で、停止することを確認しておいてください。

なお、startup.bat を実行すると、内部で shutdown.bat を call しています。
(startup.bat は、再起動:リスタート も兼ねています。)
そのため、初めて Tomcat を起動する場合は、下記のエラーが発生しますが問題ありません。

```
Catalina.stop: java.net.ConnectException: Connection refused: connect
java.net.ConnectException: Connection refused: connect
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        ・・・・・・・・
```

$UAP¥bin¥workdelete.bat の動作確認も行っておくことを推奨いたします。
これにより、旧(開発環境)でコンパイルされた JSP のクラスファイルが、
すべて削除されます。クラスファイルの整合性不一致等の発生を防止できます。
ただし、初めてアクセスされる画面は、コンパイル時間が必要なため、
画面が表示されるまで、遅くなりますので、あらかじめすべての画面にアクセスしてコンパイルしておくか、ユーザーに説明を行い、納得していただいておく必要があ

ります。

10. ブラウザでの動作確認

http://サーバー名:8080/プロジェクトID/jsp/index.jsp でアクセスして、
起動すれば、正常です。

第II部　Webアプリケーション構造

『何かを学ぶためには、自分で体験する以上にいい方法はない。』
Albert Einstein （アルベルト・アインシュタイン）

ここでは、Webアプリケーションの構造について説明します。
構成は次のとおりです。

第 2 章　ユーザーインターフェース（業務ロジックA）
JSPを用いてユーザーインターフェースを構築する場合の考え方について説明します。

第 3 章　ビジネスロジック（業務ロジックB）
業務に特化した機能をPL／SQLを用いてビジネスロジックを構築する場合の考え方について説明します。

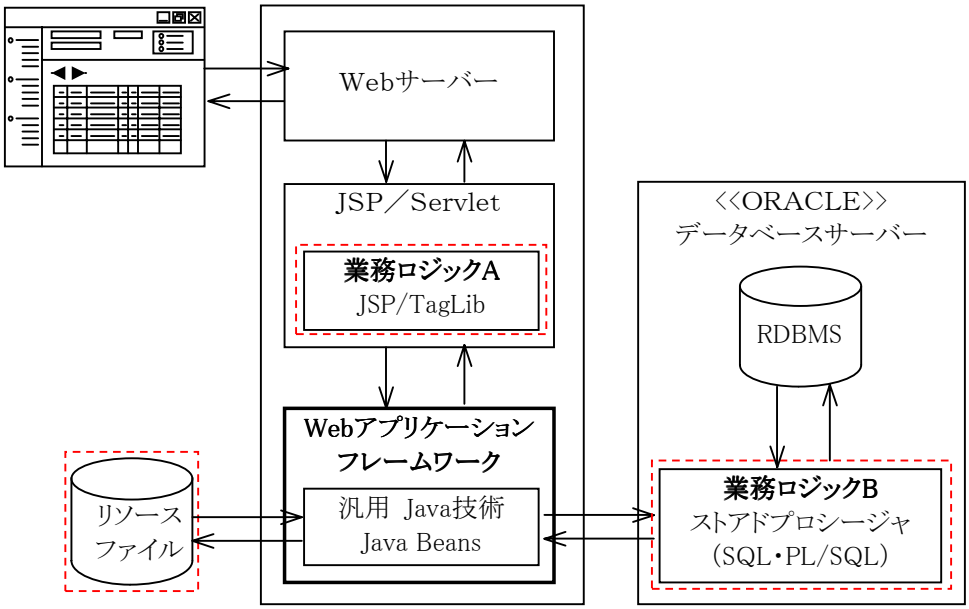
第 4 章　エンジンコア（汎用オブジェクト）
エンジンコア、特にカスタムタグと汎用オブジェクトについて、説明します。

第2章　ユーザーインターフェース（業務ロジックA）

この章では、JSPを用いてユーザーインターフェースを構築する場合の考え方について説明します。

Web アプリケーションは、3つの階層で構築されています。

- **業務ロジック A（JSP,SQL 等の画面系）**
- **業務ロジック B（PL/SQL でのバッチ系）**
- **フレームワーク（Java によるエンジン機能）**



1．業務ロジック A

JSP のタグライブラリを利用して、画面を作成します。画面遷移や、SQL、ストアドプロシージャ呼び出し等の機能も、タグライブラリを通して実行できます。この業務ロジックは、ユーザーインターフェース部に相当しますので、簡単に開発/修正できることを特長にしています。

2．業務ロジック B

オラクルのストアドプロシージャ/ストアドファンクションによる業務ロジックを記述します。　主として複雑なバッチ処理的な業務ロジックを記述し、業務ロジック A の JSP より、CALL で呼び出して利用します。この業務ロジックは、システムの基幹処理を中心に通常ユーザーインターフェースに影響されないような業務を記述します。共通的な処理を PL/SQL で記述することで、画面系からのリアルタイムな使用だけでなく、バッチ的な処理にも利用可能です。

３． Web アプリケーションフレームワーク

先の2つと異なり、システム開発者が作成するのではなく、エンジンチームと呼ばれているメンバーで、業務ロジックとは切り離されて開発/運用されています。この部分は、拡張性やメンテナンス性を重視したオブジェクト指向言語 Java で開発されています。

エンジンと業務ロジックAとを切り分けるインターフェースの役割を行うのが、リソースファイルです。当フレームワークを利用して開発する場合、各種リソースが必要になります。

リソースとは、画面、項目、メッセージ、項目選択肢、ユーザーなどの基本情報と、それに付随するアクセス制限、言語（国際化対応）、表示形式、データ論理属性など、プログラミング時にアプリケーション全般に共通に使用される情報を、ここですべて管理し、ノン・コーディングで利用できるようにします。

これらのリソースを作成するにあたり、非常に重要なのがデータベース設計です。データベース設計をするためのデータベースをあらかじめ準備しておき、その情報よりリソースデータの自動作成を行うことで,作業効率の向上を図っています。

これらのリソースファイルを基本設計時に用意しておく必要があります。この情報を用いて、画面サンプル（納入仕様書）の作成が可能になります。（物理データベースは不要。リソースのみで動作させることができます。）

リソースファイルについては、『第8章 Webアプリケーション・リソースの管理』 にて、再度取り上げたいと思います。

第3章　ビジネスロジック（業務ロジックB）

この章では、業務に特化した機能をPL／SQLを用いてビジネスロジックを構築する場合の考え方について説明します。

Webアプリケーションの考え方として、高品質、短工数、短納期　つまり、QCD を高めるという目標以外に、短変更、多流用、省管理 を挙げています。

先の業務ロジック A といわれる JSP 部分は、短変更という目標を担っています。つまり、お客様からの要望事項に合わせてすぐに変更できるためには、JSP でお手軽に開発/修正できる必要があります。
反面、この業務ロジック B では、多流用 の目標を担っているといえます。
ビジネスロジクといえども、画面や新機能は年々変化したり、担当者ごとに操作方法やその目的が異なることがあります。しかし、その企業で脈々と行われてきた処理（基幹処理）は、そう簡単には変わりません。
Pl/SQL により、旧来のビジネスロジックを構築したり、コアの部分を構築しておき、付加項目については、JSP 等で検索できる様にする事が可能なため、業務ロジックの流用を促進する事が可能になります。

第4章　エンジンコア（汎用オブジェクト）

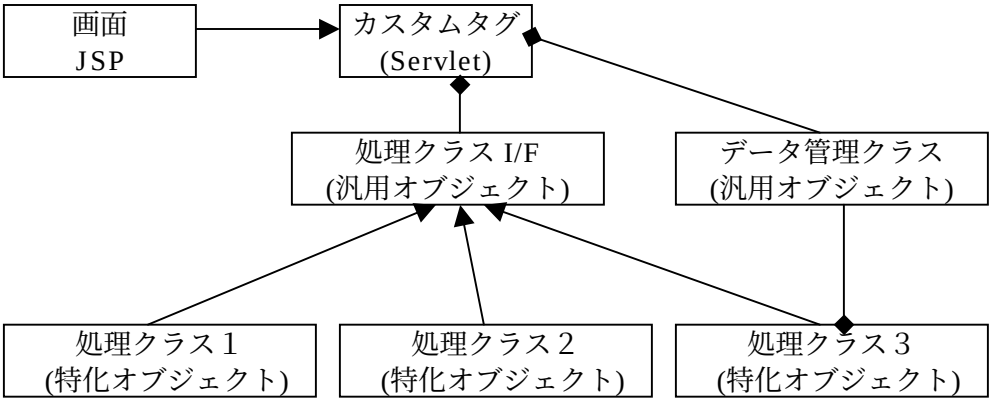
この章では、エンジンコア、特にカスタムタグと汎用オブジェクトについて、説明します。

エンジンコア部は、JSP のカスタムタグと、汎用オブジェクトから成り立っています。
JSP 画面には、カスタムタグで記述します。そのカスタムタグは、いわば、画面とエンジンコア部のインターフェースになっており、エンジンコア部を修正したとしても、JSP 画面には何ら影響を及ぼしません。
また、カスタムタグは、XML 形式のため、属性の追加時でも、既存のカスタムタグを修正することなく、（つまり、上位互換を保ったまま） エンジンコア部分を進化させることが可能になります。

カスタムタグ自身の機能を、以下に示します。

- JSP 画面からの設定情報の取得
- 設定情報に応じた汎用オブジェクトの作成/取得
- 汎用オブジェクトへ設定情報を転送
- 汎用オブジェクトより、処理結果の受取
- 処理結果の表示/または、設定

多くの場合、カスタムタグは、自分では処理を行いません。JSP や Servlet 関係のクラスを一旦汎用オブジェクト（データ管理クラス）に設定し、そこで、汎用オブジェクト（データ処理クラス）で処理を行います。結果も、汎用オブジェクト（データ管理クラス）で受け取りますので、最終表示は、これを処理するオブジェクトへの転送で実現しています。



第 III 部　Webアプリケーション アーキテクチャ

『「偶然」は、準備のできていない人を助けない。 』
パストゥール

ここでは、Webアプリケーションのアーキテクチャについて説明します。
構成は、次のとおりです。

第 5 章　Webアプリケーションの起動と停止
Webアプリケーションについて説明し、Webアプリケーション管理者がWebアプリケーションへのアクセスを制御する方法について説明します。

第 6 章　アプリケーション・アーキテクチャ
アプリケーションの構造とその実行アーキテクチャについて説明します。

第 7 章　メモリー・アーキテクチャ
Webアプリケーションで使用するメモリー構造について説明します。検索データや、リソース等のキャッシュ方法について説明します。

第 8 章　Webアプリケーション・リソースの管理
Webアプリケーションで必要なリソースを制御する方法について説明します。

第5章 Webアプリケーションの起動と停止

この章では、Webアプリケーションについて説明し、Webアプリケーション管理者がWebアプリケーションへのアクセスを制御する方法について説明します。

Tomcat サーバーを使用する場合の方法を以下に示します。

ヒント

【事前準備】

APPS¥bin¥ 以下に、 init.bat , startup.bat , shutdown.bat , workdelete.bat のファイルをインストール時に設定しておく必要があります。
さらに、init.bat の除く3ファイルのショートカットを作成しておくと、便利です。

1． init.bat

この Web アプリケーションの基本となる情報を設定します。

set APPS=H: Java,Tomcat 等のインストールドライブを指定します。
set UAP=G: Web アプリケーションのドライブを指定します。

set JAVA_HOME=%APPS%¥java¥jdk14 JAVA_HOME の指定
set CATALINA_HOME=%APPS%¥java¥tomcat TOMCAT のホームの指定
set CATALINA_OPTS=-server -Xms32m -Xmx64m -Xss256k
起動時オプションを指定します。これは、JDK 起動時オプションです。

set PATH=%JAVA_HOME%¥bin;%PATH%; 実行パスの指定

起動時オプションの指定で、Web アプリケーションで使用できるメモリサイズを設定します。
詳細は、JDK のヘルプかマニュアルをご参照ください。

2． startup.bat

Web アプリケーションを起動します。
すでに、起動済みの場合は、それを一旦終了させてから、再起動を行います。
Tomcat の状況によっては、起動済みのアプリケーションが終了しない場合があるため、,shutdown.bat を行って、完全に終了した後に再起動する方がベターです。
なお、Web アプリケーションが立上がっていない状態で、startup.bat を実行すると、しシャットダウンできないためのエラーが発生しますが、問題ありません。

3． shutdown.bat

Web アプリケーションを安全にシャットダウンします。
シャットダウン時処理で、データベースとのセッション切断や、統計情報の設定、ユー

ザー情報の記憶等を行っている場合がありますので、通常は、shutdown.bat の実行により終了させてください。
なお、startup.bat 時の終了後再起動時も、同様に安全です。

4. workdelete.bat

Web アプリケーションを再起動します。
startup.bat との違いは、JSP のコンパイルされたクラスファイルを破棄してから再起動を行います。
通常の Tomcat では、%CATALINA_HOME%¥work 以下に、JSP から、JSP クラスを生成し、これをコンパイルして使用します。このクラスを一旦削除することにより、JSP 画面の変更が確実に、Tomcat に反映されます。

コラム JSP のコンパイル

JSP ファイルを修正した場合、Tomcat はその修正を自動認識して、JSP クラスのソースを自動生成し、コンパイルします。
そのため、JSP 画面変更後の初めてのアクセスは、コンパイル時間がかかるため、通常より遅くなります。
Tomcat は、この JSP クラスと JSP 画面のタイムスタンプとを比較して、JSP クラスが JSP 画面よりも新しい場合は、再度ソース作成⇒コンパイルを行います。
ただし、このタイムスタンプの比較では,

- JSP 画面にインクルードされている画面の修正分の自動判断。
- JSP 画面の編集端末(クライアント)と JSP クラス作成端末(サーバー)間でのシステム時刻の違いによるタイミングのずれ。

が、問題になります。

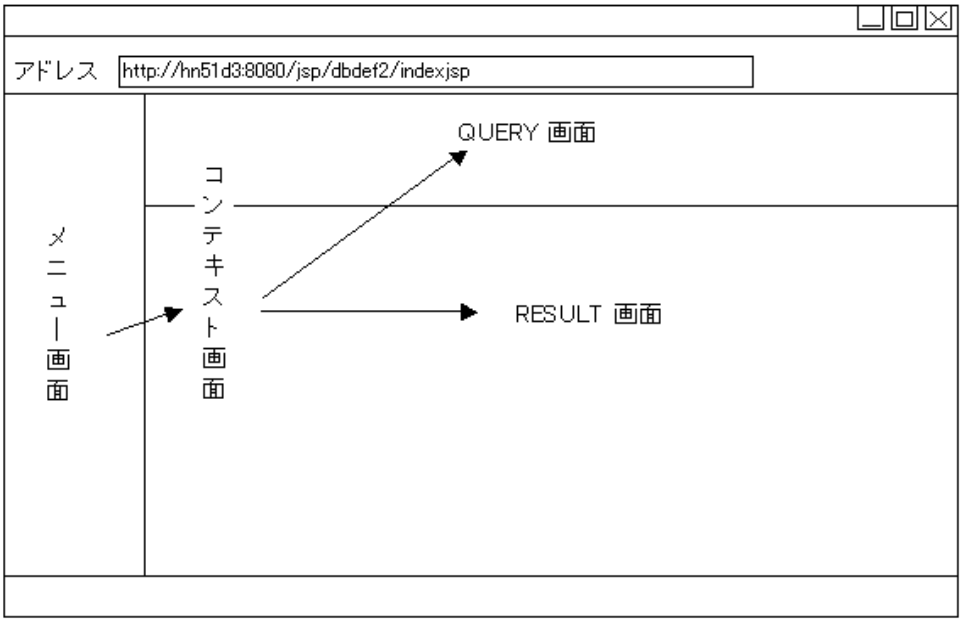
JSP のインクルード画面を修正した場合は、まったく反映されないため、比較的容易に気付きます。通常はインクルードされるファイルは、共有されている場合が多いので、workdelete.bat にて、ワークを削除した方が無難です。
サーバーとクライアントのシステム時刻がずれている場合は、非常にわかりにくい現象になります。
サーバーが、仮に2分進んでいる場合、JSP 画面を修正したあと、1回目は、すぐに変更が反映されますが、その直後(例えば2分以内)に変更しても、サーバー側が2分進んでいるため、JSP ソースとの差が(クライアント側の方が古いため)コンパイルされません。セーブできていないと思い込み、何度かセーブを繰り返しているうちに、2分が経過して、反映されます。
このケースで、毎回、workdelete.bat で反映させていたのでは時間の無駄になるため、サーバーとクライアントの時刻は、合わせておく必要があります。
そのような時刻設定サーバーを使用できない環境では、クライアント側を進めておくくらいの方が無難かもしれません。

第6章 アプリケーション・アーキテクチャ

この章では、アプリケーションの構造とその実行アーキテクチャについて説明します。

JSP 画面は、大きく、メニュー画面、QUERY 画面、RESULT 画面に分かれています。

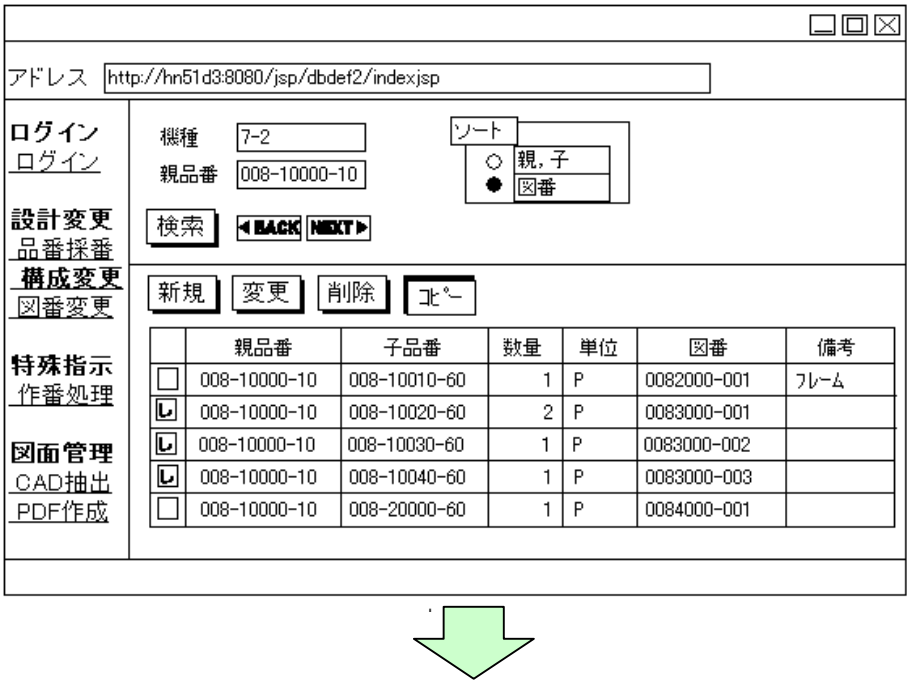
メニュー画面：
　メニューを表示します。
　基本的には,ここのメニューをクリックすると,右側のコンテキスト画面が呼ばれます。
コンテキスト画面：
　各機能ごとのフォルダは、このコンテキスト内で独自にフレーム分割を行います。
　通常は QUERY 画面, RESULT 画面をフレーム分割して使用します。
QUERY 画面：
　検索条件を登録したり,表示順を指定しています。
　この画面の『検索』ボタンで,RESULT 画面が呼ばれる。
RESULT 画面：
　検索条件を受け取り,実際に検索／表示します。
　この画面から,登録処理メニューを選ぶと,同じターゲットに登録画面が現れます。

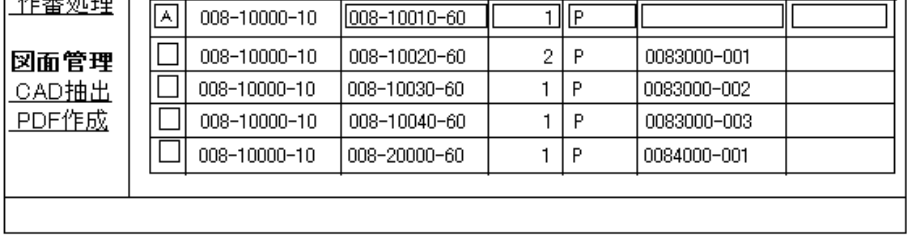


1. 画面制御

①コピー画面

コピー処理は検索画面で条件を入力(または, 何らかの業務ロジックを実行)して,その結果を RESULT 画面に結果を出力します。検索結果の画面より,新規追加／変更／削除／コピー を選ぶ事により,それぞれの処理を行います。
ユーザーがその画面の登録権限が無い場合は,編集ボタンは現れません。
論理キーの重複チェックは行っていないので、各業務アプリケーション側で対応する必要があります。





2. 画面制御

②変更画面

変更ボタンは,その項目の変更を行います。
なお，ここでの変更は,物理的なフレームワーク上のユニークキーに対しての変更であり,論理的なキー（親子など）に対しては,各業務ロジックで作り込む必要が有ります。例えば,自動的に開始／終了でＡデータとＤデータを作成するとか，子品番の項目を書き換えられない様に,チェックする（テキストフィールドにしない）などです。

③削除画面

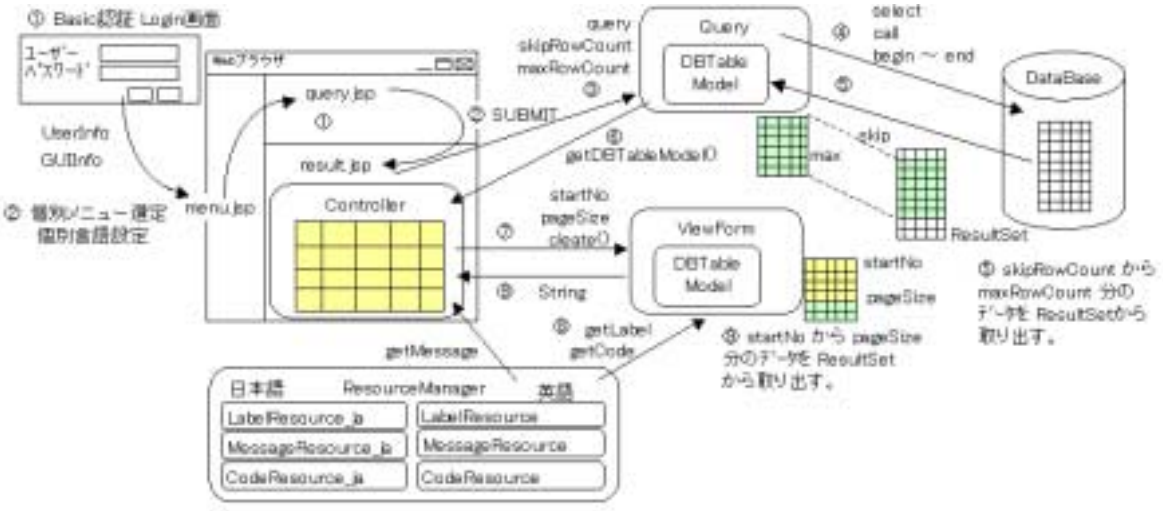
削除処理は,選ばれた項目をそのまま変更しないで,削除します。
削除する場合でも、物理削除と論理削除がありますので、それぞれの処理は，書く業務ロジックで作りこむ必要があります。

④新規追加画面

基本的には，コピー画面と同様の動きをします。
違いは、行選択時のカラムデータが、コピーは、選択された行データがセットされ、新規は、空白で行データが作成されます。
カラムセットタグのデフォルト値が異なると考えてかまいません。

第7章 メモリー・アーキテクチャ

この章では、Webアプリケーションで使用するメモリー構造について説明します。検索データや、リソース等のキャッシュ方法について説明します。



まず、通常の Web プリケーションのデータの流れを以下に示します。

① Basic 認証により Login 画面を表示（ユーザー情報取得）
② ユーザーごとにアクセスできる画面を，指定言語で表示する。
③ query.jsp より検索条件等を入力する。
④ SUBMIT することにより、result.jsp を呼び出す。以下，result.jsp 内部の処理
⑤ Query クラスを作成する。検索条件と 取り込み開始ポイント、最大取り込み可能数をセットする。
⑥ jdbc接続により、DataBase を検索する。（DB 以外にフラットファイル、XML,CSV ファイル等の取り込みも可能）
⑦ 検索結果を元に、内部に DBTableModel を生成させる。
⑧ Query より DBTableModel を取り出す。
⑨ DisplayForm を作成して内部に DBTableModel をセットする。（startNo,PageSize,lang）
⑩ ResourceManager を用いて,DBTableModel のラベル、コードを lang により指定された言語に変換する。
⑪ 変換結果の String（Stream・Writer）を返す。

1. データベース検索結果に関するメモリ管理

データベースから検索したデータは、DBTableModel オブジェクトに格納されます。
これは、セッション情報に識別子ごとにメモリに格納されます。
通常、1ユーザーごとに1セッションが割り当てられ、かつ、識別子に初期値を使用すると、
1ユーザーごとに、1つのDBTableModel を検索ごとに作成して置き換えていることになります。
また、このメモリ情報は、どの画面で検索した結果でもいつでも取り出すことができます。

不要なメモリを残さないためには、識別子は、デフォルトのみとするか、または、別に使用した場合は、すぐに破棄するようにしなければなりません。
また、DBTableModel にセットするデータ量も制限することで、多くのログインユーザーがいてもメモリ不足にならないようにする必要があります。

以下に query タグのメモリ制御に関係する属性を説明します。

- tableId
  メモリに格納する場合の識別子です。通常はデフォルト(指定しない)を使用します。
- scope
  セッションやリクエストなど、どのスコープに保存するか指定します。
  通常はデフォルト(指定しない)ですが、検索のみなど、tableId を指定して登録系と別に結果を表示させたい場合に利用できます。
  ただし、リクエストスコープに登録した場合は、データ抜き出しにより取り出すことはできません。
- maxRowCount
  検索時の最大行数を指定します。
  不適切な検索絞込み条件によって、大量の検索結果によりメモリ不足を引き起こさないために、最大件数を制限できます。
  デフォルトは、SystemResource.properties の DB_MAX_ROW_COUNT で指定できます。

2. リソース情報のメモリ管理

リソース情報とは、ユーザー、画面、ラベル、カラム、メッセージなどの情報で、言語(日本語、英語、中国語等)ごとに存在しています。
これらは、セッション情報ではなく、アプリケーションごとに共通で使用します。
つまり、ログインユーザーが何人でも、使用するメモリ量は変わりません。
ただし、1つのTomcatで立ち上げるWebアプリケーション分だけ、必要になります。

リソース情報については,『第8章 Webアプリケーション・リソースの管理』で、詳細を説明します。

3. Tomcat起動時のJavaオプション

Tomcat起動時に、Javaオプションを指定します。
このオプションで、Java実行時のメモリ割り当てを指定できます。
オプションは、APPS¥bin¥init.bat に、記述しています、CATALINA_OPTS で指定します。

例）
set CATALINA_OPTS=-server -Xms32m -Xmx64m -Xss256k

-server
Java HotSpot Server VM を選択します。

-Xmsn
メモリ割り当てプールの初期サイズをバイト数で指定します。
指定する値は、1M バイトより大きい 1024 の倍数にしなければなりません。
キロバイトを指定するには、文字 k または K を付けます。メガバイトを指定するには、
文字 m または M を付けます。 既定値は 2M バイトです。 次に例を示します。
-Xms6291456
-Xms6144k
-Xms6m

-Xmxn
メモリ割り当てプールの最大サイズをバイト数で指定します。
指定する値は、2M バイトより大きい 1024 の倍数にしなければなりません。
キロバイトを指定するには、文字 k または K を付けます。メガバイトを指定するには、
文字 m または M を付けます。 既定値は 64M バイトです。 次に例を示します。
-Xmx83886080
-Xmx81920k
-Xmx80m

参照:
JavaTM 2 SDK, Standard Edition
ドキュメント
V1.4.0

api14/tooldocs/win32/java.html

第 IV 部　データ制御オブジェクト

『この素晴らしい応用科学は労働を軽減し、生活をより豊かにしながら、
なぜ我々に幸福をもたらしてくれないのか。答えは簡単である。
我々がそれを有意義に利用するにいたっていないからである。』
Albert Einstein （アルベルト・アインシュタイン）

ここでは、データの制御について説明します。
構成は、次のとおりです。

第 8 章　カラムオブジェクト
データベースのカラム属性に対するオブジェクトについて、説明します。

第 9 章　コードリソースとカラムオブジェクト
カラムオブジェクトを修飾する機能として、コードリソースがあります。コードリソースとカラムオブジェクトの関係を説明します。

第 10 章　レンデラー、エディター、DBタイプ
カラムオブジェクトに対して、データを適用するときに、表示（レンデラー）、編集（エディター）、物理特性（DBタイプ）の取扱を説明します。

第 11 章　その他リソースファイル
ラベル、メッセージ、ユーザー、画面、の各リソースについて説明します。

第8章 カラムオブジェクト

この章では、データベースのカラム属性に対するオブジェクトについて、説明します。

本，フレームワークで最も重要な考えが，このカラムオブジェクトです。
通常、データベースから検索した結果は、文字列(VARCHAR2)や数字(NUMBER)などのプリミティブ型情報しか持っていません。もちろん，桁数などの情報は持っていますが、それ以上の情報（例えば、全角/半角や、大文字/小文字、その他ユーザーカスタマイズ情報）をコントロールすることはできません。

カラムオブジェクトとは、カラム名に対応するオブジェクトを割り当て、検索結果に適用させることでオブジェクト的な振る舞いを検索結果に持たせることを可能にするオブジェクトのことです。

カラムオブジェクトで指定できる属性は、以下のとおりです。

- カラム名
- カラムのクラスを文字列にした名称
- カラムの文字桁数定義
- カラムが書き込み可能かどうか
- 表示用レンデラー(パラメーター)
- 編集用エディター(パラメーター)
- データのタイプ（パラメーター）
- データのデフォルト値
- カラムのラベル ID
- カラムのコード ID

・カラム名

カラムオブジェクトのキーになります。
カラムオブジェクトは、検索結果のカラム名をキーにオブジェクトを構築します。
また、カラム名は、データベースのテーブルにまたがって，同一名称が使われることがあります。
逆にいえば、カラムオブジェクトを決定するカラム定義は、1システム（Webアプリケーション）を通して、ユニークである必要があります。テーブル間にまたがる場合でも、同一カラム名は、同一カラムオブジェクトに割り当てられます。

・カラムのクラスを文字列にした名称

VARCHAR2 や NUMBER といった、カラムのデータベース属性を指定します。これは、将来的な拡張を考慮していますが、現時点では、この2種類のみサポートしています。
HTML 出力時に、CSS（Cascading Style Sheets）のクラス属性としてスタイルを指定できます。
デフォルトでは、VARCHAR2 は左詰、NUMBER は右詰で表示されます。

例：default.css ファイルでの記述

jsp/common/default.css

```
.VERCHER2 ,.VERCHER ,.CHER {
    text-align:         left;
    vertical-align: middle;
}

.NUMBER {
    text-align:         right;
    vertical-align: middle;
}
```

・カラムの文字桁数定義

カラムの桁数を指定します。（文字数ではありません。）
この値に応じて、テキストフィールドならばその大きさと入力文字数制限を行います。この値は、編集用エディターや、データのタイプに応じて、最適な設定が行われます。
また、少数点表記などの整数部、小数部の桁数指定にも使用されます。

・カラムが書き込み可能かどうか

現在使われていません。

・表示用レンデラー(パラメーター)

カラムの値をどのように表示するか、指定します。
例えば、数字なら、カンマ編集したり、金額なら円マークをつけたり、日付なら年/月/日に加工したりできます。
mis.pdm.hayabusa.db.column. DBCellRenderer_XXX.java の XXX 部を指定します。
(パラメーター)には、このレンデラーに対する引数を指定できます。
詳細は、第11章 レンデラー、エディター、DBタイプ にて、述べます。

・編集用エディター(パラメーター)

カラムの値をどのように編集するか、指定します。
テキストフィールドやプルダウンメニューなど、値のセットの方法を指定します。
mis.pdm.hayabusa.db.column. DBCellEditor_XXX.java の XXX 部を指定します。
(パラメーター)には、このエディターに対する引数を指定できます。
詳細は、第11章 レンデラー、エディター、DBタイプ にて、述べます。

・データのタイプ（パラメーター）

カラムの値の属性を決定します。
この属性に応じて、値の整合性をチェックします。
例えば、数字タイプであれば、それに使用できる文字やルール（カンマや少数点の位置、マイナス符号の位置など）をチェックします。

mis.pdm.hayabusa.db.column. DBType_XXX.java の XXX 部を指定します。
(パラメーター)には、このタイプに対する引数を指定できます。
詳細は、第11章 レンデラー、エディター、DBタイプ にて、述べます。

・データのデフォルト値

カラム自身のデフォルト値を指定します。
これは、アプリケーション全体で共通に使用されます。通常はデータベースのテーブルごとに初期値が変わる可能性がありますので、そのようなケースには適用できませんので、ご注意願います。

・カラムのラベル ID

カラムオブジェクト自身は、国別リソースではありません。(つまり、共通属性)
しかし、内部的にラベル ID によって、国別ラベルを生成して管理しています。
つまり、オブジェクト自身は、国別に生成されます。
ラベル ID が指定されていない場合は、カラム ID が使用されます。

・カラムのコード ID

カラムのコード ID も、ラベル ID と同様です。
コード ID が指定されていない場合は、カラム ID が使用されます。

第9章 コードリソースとカラムオブジェクト

この章では、カラムオブジェクトを修飾する機能として、コードリソースがあります。コードリソースとカラムオブジェクトの関係を説明します。

コードリソースは、キー(コードID)と選択枝からなります。
選択枝は、選択値 選択ラベル という組を複数指定することができます。

例:
コードID=選択値1 選択ラベル1 選択値2 選択ラベル2 選択値3 選択ラベル3 ・・・

この、コードリソースを、カラムリソースの、レンデラー、エディターと組み合わせることで、検索結果から、プルダウンメニューや、選択値に対応するラベルを自動作成する事ができます。
コードリソースは、ラベルリソースと同様に、国別に複数のリソースを持つことができ, かつ同時に使用することが可能なため、同一アプリケーションによる国際化対応が可能になります

表示用レンデラーが、MENU の場合、データベースの検索結果の値と、コードリソースの選択値が比較され、合致した選択ラベルが表示されます。
これにより、選択値が、1、2、3や A,B,C などの記号(コード)の場合でも, その記号に応じた表示をさせることが可能になります。(例えば、1⇒要求、2⇒受付、3⇒完成など)
また、編集用エディターでは、これをプルダウンメニューで表示できます。ユーザーは、ラベルで選択できるため、非常に判りやすくなります。
また、コードリソースに使用可能な選択値を指定しておくことより、データベース定義をより完全に定義することが可能になります。(つまり、DB 設計時点の情報が、そのままノンコーディングで利用できます。)
さらに、SEQMENU などの編集用エディターを用いれば、1⇒2⇒3 というように、一度設定した値以降しか選択できない(先の例では, 受付後に、要求に戻せない)などの制御を、自由に行うことが可能になります。

第10章 レンデラー、エディター、DBタイプ

この章では、カラムオブジェクトに対して、データを適用するときに、表示（レンデラー）、編集（エディター）、物理特性（DBタイプ）の取扱を説明します。

1. 表示用レンデラー

表示用レンデラーは、カラムの値をどのように表示するか、指定します。
例えば、数字なら、カンマ編集したり、金額なら円マークをつけたり、日付なら年/月/日に加工したりできます。
mis.pdm.hayabusa.db.column. DBCellRenderer_XXX.java の XXX 部を指定します。
カラムリソースの構築時にパラメーターとして引数を指定できます。

表示用レンデラーには、以下のクラスが用意されています。

レンデラー	説明
COLUMN	カラムのデータを表示する場合に使用するクラスです。動的カラムを生成する場合に使用します。
FORM	パラメータで指定された FORM を表示するクラスです。元の Value を、$1 として、使用することが可能です。
HMS	カラムのデータを時:分:秒に分けて表示する場合に使用するクラスです。
HTML	HTML タグを含むデータを表示する場合に使用するクラスです。クロスサイトスクリプティング問題に対応するフィールドに対して定義することにより、エスケープ処理を行います。
LABEL	カラムのデータをそのまま文字列として表示する場合に使用するクラスです。
MENU	カラムのデータをコードリソースに対応したラベルで表示する場合に使用するクラスです。
MONEY	カラムのデータを金額表示する場合に使用するクラスです。
NUMBER	カラムのデータを数字表示する場合に使用するクラスです。
PASSWD	カラムのデータをパスワード情報として表示する場合に使用するクラスです。パスワード表示とは、すべての表示を、****** で表示します。
PN	カラムのデータを品番情報として表示する場合に使用するクラスです。これは、特殊な表示で、11桁の文字列を3-5-3表示します。
PRE	カラムのデータをそのまま PRE 表示する場合に使用するクラスです。
QUERY	パラメータで指定された SQL 文を実行し、その結果を表示するクラスです。元の Value を、$1 として、使用することが可能です。
YM	カラムのデータを日付（年/月）表示する場合に使用するクラスです。
YMD	カラムのデータを日付（年/月/日）表示する場合に使用するクラスです。
YMDH	カラムのデータを日時（年/月/日 時:分:秒）表示する場合に使用するクラスです。

2. 編集用エディター

編集用エディターは、カラムの値をどのように編集するか、指定します。

テキストフィールドやプルダウンメニューなど、値のセットの方法を指定します。
mis.pdm.hayabusa.db.column. DBCellEditor_XXX.java の XXX 部を指定します。
カラムリソースの構築時にパラメーターとして引数を指定できます。

編集用エディターには、以下のクラスが用意されています。

エディター	説明
COLUMN	カラムのデータを編集する場合に使用するクラスです。動的カラムを生成する場合に使用します。
HTML	HTML タグを含むデータを編集する場合に使用するクラスです。クロスサイトスクリプティング問題に対応するフィールドに対して定義することにより、エスケープ処理を行います。
MENU	カラムのデータをコードリソースに対応したプルダウンメニューで編集する場合に使用するクラスです。
NUMBER	カラムのデータを数字編集する場合に使用するクラスです。
PASSWD	カラムのデータをパスワード情報として編集する場合に使用するクラスです。パスワード編集とは、すべての入力値を、****** に置き換えて表示します。
SEQMENU	このメニューは、正方向にしか選べません。指定した値より小さな値は、メニューとして表示しないようにします。これにより、シーケンスにステータスを順に挙げていくような、メニューを作成することができます。（逆に戻れないメニュー）
TEXT	カラムのデータをテキストフィールドで編集する場合に使用するクラスです。
TEXTAREA	カラムのデータをテキストエリアで編集する場合に使用するクラスです。
YM	カラムのデータを日付（年/月）編集する場合に使用するクラスです。
YMD	カラムのデータを日付（年/月/日）編集する場合に使用するクラスです。
YMDH	カラムのデータを日時（年/月/日 時:分:秒）編集する場合に使用するクラスです。

3. データのタイプ

データのタイプは、カラムの値の属性を決定します。
この属性に応じて、値の整合性をチェックします。
例えば、数字タイプであれば、それに使用できる文字やルール（カンマや少数点の位置、マイナス符号の位置など）をチェックします。
mis.pdm.hayabusa.db.column. DBType_XXX.java の XXX 部を指定します。
カラムリソースの構築時にパラメーターとして引数を指定できます。

データのタイプには、以下のクラスが用意されています。

DB タイプ	説明
K	全角のみの Char または Varchar2 属性に対応するクラスです。
MD5	MessageDigest により、MD5 でハッシュした文字を規定するオブジェクトです。
PN	品番情報のカラム属性を規定するオブジェクトです。3-5-3品番情報を判定します。また、それぞれのフィールドにおいて、使用可能文字（例えば、Rev 文字列の制限）などを考慮しています。
R	少数点数字型文字列のカラム属性を規定するオブジェクトです。半角数字の NUMBER 属性に対応するクラスで '0' ～ '9','-','.'でのみ構成されている数字型文字列カラムです。さらに、カンマ','が含まれていても OK とします。ただし、データからは取り除きます。
S9	数字型文字列のカラム属性を規定するオブジェクトです。半角数字の NUMBER 属性に対応するクラスで '0' ～ '9','-' でのみ構成されている数字型文字列カラムです。さらに、カンマ','が含まれていても OK とします。ただし、データからは取り除きます。
X	半角の Char または Varchar2 属性に対応するクラスで一般的な制限のない文字列カラムです。
X9	数字型文字列のカラム属性を規定するオブジェクトです。半角数字の NUMBER 属性に対応するクラスで '0' ～ '9','-' でのみ構成されている数字型文字列カラムです。さらに、カンマ','が含まれていても OK とします。ただし、データからは取り除きます。
XH	半角文字＋半角カタカナに対応するクラスで、大文字小文字の一般的な制限のない文字列カラムです。
XHU	半角文字＋半角カタカナに対応するクラスで、大文字のみに制限された文字列カラムです。ホスト送信用なので、半角カタカナ小文字は使用できません。
XK	半角/全角混在の Char または Varchar2 属性に対応するクラスで一般的な制限のない文字列カラムです。
XL	半角小文字の英数字の CHAR または VARCHAR2 属性に対応するクラスです。半角カタカナは対象外です。
XU	半角大文字の英数字の CHAR または VARCHAR2 属性に対応するクラスです。半角カタカナは対象外です。

DB タイプ	説明
YM01	文字列の開始日付属性のカラムを規定するオブジェクトです。半角の Char または Varchar2 属性に対応するクラスで YYYYMM01 または、00000000 , 99999999 を許可します。ただし、日付としての整合性チェックは行いません。
YM31	文字列の終了日付属性のカラムを規定するオブジェクトです。半角の Char または Varchar2 属性に対応するクラスで YYYYMM31 または、00000000 , 99999999 を許可します。ただし、日付としての整合性チェックは行いません。
YMD	文字列の日付属性(年/月/日)のカラムを規定するオブジェクトです。半角の Char または Varchar2 属性に対応するクラスでYYYYMMDD に対応している必要があります。ただし、日付の整合性チェックは行いませんが、valueAdd(String value)による日付の加算時には正式な日付で加算されます。
YMDH	文字列の日付属性(年/月/日 時:分:秒)のカラムを規定するオブジェクトです。半角の Char または Varchar2 属性に対応するクラスでYYYYMMDDHHMMSS に対応している必要があります。ただし、日付の整合性チェックは行いませんが、valueAdd(String value)での日付の加算時には、正式な日付データにて加算します。

コラム　レンデラー、エディター、DBタイプ

レンデラー、エディター、DBタイプのクラスが、現時点ではエンジン内部でのみ定義可能になっていますが、次期バージョンでは、ユーザー定義可能なようにします。これにより、データベースのカラムを自由にオブジェクト化することができ、詳細なチェックや表示方法、編集方法を、アプリケーション共通で取り入れることが可能になります。

また、汎用的と認められたクラスをエンジン標準として取り入れることで、標準化、共通化が促進されると考えています。

第11章 その他リソースファイル

この章では、ラベル、メッセージ、ユーザー、画面、の各リソースについて説明します。

1. ラベルリソース

ラベルリソースは、キー（ラベルID）と値（ラベル値）からなります。
キーは、カラムIDと連動しており、カラムに対して表示ラベルを設定するイメージになります。
ラベルリソースは、国別に複数のリソースを持つことができ，かつ同時に使用することが可能なため、同一アプリケーションによる国際化対応が可能になります。
国別リソースの切替は，ユーザー情報の言語で指定できますので、ログインする端末やIPアドレス等による切替でないため、使用者がどこに居ても自分の言語で表示することが可能になります。

2. メッセージリソース

メッセージリソースは、キー（メッセージID）とメッセージからなります。
メッセージリソースは、ラベルリソースと同様に、国別に複数のリソースを持つことができ，かつ同時に使用することが可能なため、同一アプリケーションによる国際化対応が可能になります。
メッセージリソースには、引数を渡すことが可能です。これは、java.text.MessageFormatクラスと同等の使用方法が可能です。ただし、引数には、String（文字列）しか渡すことはできません。

3. ユーザーリソース

ユーザーリソースは、キー（ユーザーID）とユーザー属性からなります。
ユーザー属性は、ログイン時のユーザーIDから、UserInfoオブジェクトを作成し、JSP上では、userInfo タグでアクセスすることが可能です。
ユーザーリソースは、国別にリソースを持っていません。

・ユーザー

ユーザーリソースのキーになります。ログインユーザーは、セッションごとに作成されるため、1人が同時にサーバーに接続して、別々の処理を行うことが可能になります。

・パスワード

ユーザー認証用のパスワードです。本フレームワークでは、このパスワードは使用していません。ユーザー認証には、Tomcat の認証を利用しています。

・言語

ユーザーごとに言語を持っており、この言語に応じて国別リソースが使用されます。そのため、ユーザーがどこからログインしようとも、自分の言語に応じた画面

で操作することが可能になります。

・日本語名称
母国語での氏名情報です。画面上にログインユーザーとして表示します。

・英語名称
英語での氏名情報です。基本的に、英語での氏名情報は必須項目として使用できるように登録しておいてもらいます。

・メールアドレス
メール等を利用する場合のメールアドレスを指定します。
現時点では、直接使用していません。

・メールユーザー名
メール等を利用する場合のメールユーザー名を指定します。
現時点では、直接使用していません。

・メールパスワード
メール等を利用する場合のメールパスワードを指定します。
現時点では、直接使用していません。

・ロール
ユーザーのロール（役割）を指定します。これは、画面制御用の権限を付与することが可能です。通常は，役職（部長、管理者、などの役職）を指定し、その権限に応じた画面のアクセス制御を行います。
このロールが、root の場合は、すべての画面にアクセスすることが可能です。

・ユーザーグループ
ユーザーのグループを指定します。
これは、画面制御用の権限を付与することが可能です。通常は，グループ（設計、製造、営業や、技術部、業務部など）を指定し、その権限に応じた画面のアクセス制御を行います。

・プロジェクト
ユーザーのプロジェクトを指定します。
これは、画面制御用の権限を付与することが可能です。通常は，プロジェクト（製品 A タイプ、製品 B タイプ、新商品開発チームなど）を指定し、その権限に応じた画面のアクセス制御を行います。

・IP アドレス
ユーザーのログイン端末の IP アドレスを指定します。
IP アドレスにより何らかの制限や機能を付加したい場合に利用できるように情報を持っています。

4. GUIリソース

GUIリソースは、キー（画面ID）と画面属性からなります。
画面属性は、各JSP画面のホルダ名と連動しています。※
画面属性は、GUIInfo オブジェクトより、guiInfo タグでアクセスすることが可能です。
GUI リソースとユーザーリソースの関係より、どのユーザーがどの画面をアクセスできるか設定する事が可能です。
GUIリソースは、ラベルリソースと同様に、国別に複数のリソースを持つことができ，かつ同時に使用することが可能なため、同一アプリケーションによる国際化対応が可能になります。

・画面 ID
　GUIリソースのキーになります。
　画面 ID と実行アドレスは、基本的には1対1の関係ですが、異なる画面 ID から同一実行アドレスを実行することも可能です。

・実行アドレス
　メニューから呼ばれるディレクトリです。
　一般には、画面 ID と同じです。
　http で始まるアドレスを指定した場合は、ホルダではなく、指定の HTTP アドレスにアクセスします。

・表示順
　画面の表示順を指定します。
　この表示順で、画面が上から順番に表示されます。大項目として、下記のメニュー分類でブレイクすると、次のメニュー分類のグループであるという認識をしますので、例えメニュー分類が同一でも、表示順で分断された場合は、異なるグループに割り振られますので、ご注意ください。

・メニュー分類
　メニューを大項目としてグルーピングします。
　先の表示順でも述べましたように、順番が重要です。

・画面名称
　メニューフレーム上に表示される画面名称です。
　フレームの大きさに合わせて、簡略化した名称にする事ができます。
　また、メニュー分類と組み合わせることで、短い名称であっても判りやすい名称をつけることが可能になります。

・画面名称(long)
　コンテンツフレームの各画面上部に表示される画面名称です。
　これは、文字数的に余裕があるため、判りやすい名称を記述します。

・ロール

ユーザーロールに対する権限を複数指定できます。
ユーザーのロールは、この画面のロールに含まれていないか、文字列での検索を行います。
画面側には、複数のロールを記述することで、同時に多数のロールを受け入れることが可能になります。

・ユーザーグループ

ユーザーグループに対する権限を複数指定できます。
ユーザーのグループは、この画面のグループに含まれていないか、文字列での検索を行います。
画面側には、複数のグループを記述することで、同時に多数のロールを受け入れることが可能になります。

・プロジェクト

ユーザープロジェクトに対する権限を複数指定できます。
ユーザーのプロジェクトは、この画面のプロジェクトに含まれていないか、文字列での検索を行います。
画面側には、複数のプロジェクトを記述することで、同時に多数のプロジェクトを受け入れることが可能になります。

・アクセスモード

ロール、グループ、プロジェクト、その他の各読取/書込制限を、rwrwrwrw
文字列で管理します。
'r' は、読取可能を示し、 'w' は、書込可能を示します。 r-r-r-r- では、読取専用という事になり、 -w-w-w-w では、メニューには現れない隠し画面という事になります。
前から2つづつ、rw のペアに、ロール、グループ、プロジェクト、その他という4つのカテゴリに別れて判定します。その他だけ特殊で、それ以外の条件を AND 判定した結果に、OR 判定として付け加えます。

・ターゲット

メニューフレームから指定されるリンクのターゲットを指定します。
通常は, CONTENTS を指定しておき、各画面ホルダの inde.jsp が呼ばれ、そこから、QUERY と RESULT フレームに別れます。
ここに、それら以外の名称(例えば、_new)を指定すると、ポップアップに、_top で、自分自身のフレームを入れ替えます。

第V部　データ・アクセス

『定義は、定めるよりも反論するほうがやさしい。 』
アリストテレス

ここでは、データのアクセス方法について説明します。
構成は、次のとおりです。

第 12 章　SQL 、PL/SQL および Java
データを取得するために使用する SQL（Structured Query Language ）と、手続き型言語機能拡張である PL/SQL について説明します。

第 13 章　QueryとUpdate
データの取り出し方、更新の仕方について、説明します。なお、取り出すストレージは、データベースだけではなく、ファイルやメールサーバーでも構いません。

第 14 章　View
取り出したデータの出力方法について、説明します。通常は、標準出力（画面表示）ですが、ファイルへの入出力方法についても、説明します。

第 15 章　高度なデータアクセス
通常のSQLによるデータ取り出しと、カラムオブジェクトの機能を組み合わせることによって、高速で汎用性の高いデータ取り出し方法について、説明します。

第12章 SQL 、PL/SQL および Java

この章では、データを取得するために使用する SQL（Structured Query Language ）と、手続き型言語機能拡張である PL/SQL について説明します。

1. 共通オブジェクトの定義

PL/SQL の共通オブジェクト定義は、次のとおりです。

①ARG_ARRAY

CREATE OR REPLACE TYPE ARG_ARRAY AS VARRAY(100) OF VARCHAR2(100);

・引数の受け渡しは、配列タイプで行います。
・すべて文字列タイプとし、必要であれば，PL／SQL内でNUMBER型等に変換します。
・引数の最大数は100個とし、最大文字数は、100桁とします。
・引数の順番は、query タグの args 属性で設定します。
・もちろん、受け取り側は、その順番で受け取る必要があります。

②ERR_MSG_ARRAY

```
CREATE OR REPLACE TYPE ERR_MSG AS OBJECT
  ( NO NUMBER,                    /* 行番号 */
    KEKKA OUT NUMBER ,            /* 結果  0:正常 1:警告 2:異常 */
    ID VARCHAR2(10),              /* エラーメッセージID */
    MSG1 VARCHAR2(100),           /* メッセージの引数1 */
    MSG2 VARCHAR2(100),           /* メッセージの引数2 */
        ……
    MSG5 VARCHAR2(100) );         /* メッセージの引数5 */
/

CREATE OR REPLACE TYPE ERR_MSG_ARRAY AS VARRAY(500) OF ERR_MSG;
/
```

・エラーメッセージは、オブジェクトタイプで宣言し、最大 500件まで登録することができます。
・それ以上のエラーが出た場合は、表示を打ち切ることとします。
・エラーメッセージに引数を5個まで登録できます。

③SYSARG_ARRAY

```
CREATE OR REPLACE TYPE  SYSARG AS OBJECT
  ( NO NUMBER,                    /* 行番号 */
    CDKH VARCHAR2(1) ) ;          /* 改廃コード A:追加 C:変更 D:削除 */
/
```

```
CREATE OR REPLACE TYPE  SYSARG_ARRAY
          AS VARRAY(1000) OF  SYSARG;
/
```

・プロシジャーに渡すシステム制御情報は、行番号と改廃コードとします。
・エラーメッセージや処理の振り分けは、これらの情報を用いて PL/SQL 側で記述します。
・これらは、1000件までの情報を渡しますが、それ以上の登録が画面から必要な場合は、別途専用の PL/SQL と、登録用 Java プログラムが必要になります。
・それ以上の情報の一括登録は、画面情報をインプットとするのではなく、別に、要求テーブルや、ファイルなどを利用して DB 登録する事を推奨します。

④PROCEDURE SET_ERRMSGS

```
CREATE OR REPLACE PROCEDURE SET_ERRMSGS
    (P_ERRMSGS   IN OUT   ERR_MSG_ARRAY,
     P_NO        IN       NUMBER     := NULL,
     P_KEKKA     IN       NUMBER     := NULL,
     P_ID        IN       VARCHAR2 := NULL,
     P_MSG1      IN       VARCHAR2 := NULL,
     P_MSG2      IN       VARCHAR2 := NULL,
     P_MSG3      IN       VARCHAR2 := NULL,
     P_MSG4      IN       VARCHAR2 := NULL,
     P_MSG5      IN       VARCHAR2 := NULL) IS

    TOO_MANY_ERRORS   EXCEPTION; --エラー件数最大数オーバーエラー
    V_ERRMSG          ERR_MSG;    -- エラーメッセージオブジェクト
```

エラーメッセージ配列を設定する場合の初期化などを行うプロシージャ

2. 検索系のコール条件

検索系の PLSQL の procedure のコール条件は、次のとおりです。

①検索専用 PL/SQL

条件を与えて、検索結果をカーソルで返します。
検索におけるエラーは、入力値の不正な値だけなので、基本的にはエラーメッセージは返しません。
なお、この PL/SQL 内でデータベースの更新を行うことは、PL/SQL 作成者の責任の元、行うことができます。

```
CREATE OR REPLACE PACKAGE HAYABUSA IS
    TYPE CUST_CURSOR IS REF CURSOR ;
    PROCEDURE XXXXXX (
KEKKA OUT NUMBER,
ERRMSG OUT ERR_MSG_ARRAY,
```

```
RC1     OUT CUST_CURSOR,
ARGS    IN    ARG_ARRAY );
END;
/

CREATE OR REPLACE PACKAGE BODY HAYABUSA IS
PROCEDURE XXXXXX (
KEKKA OUT NUMBER ,                  /* 結果   0:正常 1:警告 2:異常 */
ERRMSG OUT ERR_MSG_ARRAY,           /* エラーメッセージ配列 */
RC1       OUT CUST_CURSOR,          /* 結果はカーソルタイプで返す。 */
ARGS     IN     ARG_VARRAY,         /* 引数は、文字列の配列タイプで
                                        受け取る。  */
)  AS
BEGIN
      OPEN RC1 FOR
SELECT A.COLUMN_ID,A.TABLE_NAME,A.COLUMN_NAME,B.NAME_JA,
B.DATA_TYPE,B.USE_LENGTH,A.FGADD
FROM DB05 A, DB03 B
WHERE A.COLUMN_NAME = B.COLUMN_NAME
ORDER BY A.SYSTEM_ID,A.TABLE_NAME,A.COLUMN_ID ;
END XXXXXX;
END HAYABUSA;
/
```

３．登録系のコール条件

登録系の PLSQL の procedure のコール条件を以下のように定めます。

①登録専用 PL/SQL

条件を与えて、検索結果をメッセージ配列で返します。
登録においては,結果として、OK/NG を返し、NG の場合にメッセージを表示します。なお、この PL/SQL 自体で全ての処理を完結させる必要があります(トランザクション制御は、このPL/SQLで自分で対応する)。登録結果を再度検索したい場合は, JSP 画面上で queryタグを用いて再検索してください。

```
CREATE OR REPLACE PROCEDURE YYYY(
KEKKA OUT NUMBER ,                  /* 結果   0:正常 1:警告 2:異常 */
ERRMSG OUT ERR_MSG_ARRAY,           /* エラーメッセージ配列 */
NAMES            IN   VARCHAR2,     /* カラム名チェック用文字列 */
SYSARGS     IN   SYSARG_ARRAY,  /* 登録条件配列 */
USERARGS     IN   USERARG_ARRAY,        /* 登録データ配列 */
)  IS
……


CREATE OR REPLACE TYPE  USERARG AS OBJECT ( ····· );
```

```
/

CREATE OR REPLACE TYPE  USERARG_ARRAY AS VARRAY(XXX) OF  US
ERARG;
/
```

・ユーザー定義オブジェクトの配列は、各プロシージャごとに作成します。
・これは、各引数の個数等は、個別に設定することとします。
・NAMES はカラム名チェック用の文字列です。ここには、USERARGSで定義されているカラム名の順番を、カンマ区切りで設定します。
　　例 'OYA,KO,HOJO,DYSTR,DYEND'
・エンジン側は、上記のカラム名を元にデータを配列に設定しますので、受け側（PL/SQL）では、自分自身の定義とチェックするように作ってください。

第13章 QueryとUpdate

この章では、データの取り出し方、更新の仕方について、説明します。なお、取り出すストレージは、データベースだけではなく、ファイルやメールサーバーでも構いません。

query タグは、データベースからデータを検索して、DBTableModel を作成します。
同様に、update タグは、DBTableModelの値を、データベースに書き込みます。

どちらも、DBTableModel の値をやり取りしますので、query タグで検索した結果を、そのまま update タグで登録したり、ファイルから読み取った、DBTableModel をデータベースに、update したり、query で検索した DBTableModel をファイルに書き出したり相互運用可能です。

query タグも、update タグも、派生クラスがしますので、以下に説明いたします。

1. query タグ

query タグは、データベース検索に使用されるタグです。
queryType の指定により、実際に呼び出す Query クラスを指定します。
通常は、queryType = "JDBC" が良く使われますが、PL/SQL によりカーソルを返す場合は, "JDBCErrMsg" を指定します。

<mis:query
tableId : (通常は使いません)結果を DBTableModel に書き込んで、session に登録するときのキーを指定します。
queryType : 検索を実行する手段を指定します。
dbid : (通常は使いません)Query オブジェクトを作成する時の DB 接続 ID を指定します。
scope : キャッシュする場合のスコープを指定します。
command : 処理コマンドをセットします。
maxRowCount : (通常は使いません)データの最大読み込み件数を指定します。
SkipRowCount : (通常は使いません)データの読み始めの初期値を指定します。
debug : デバッグ情報(タグのボディー部)を 出力する／しないを指定します。
displayMsg : 検索結果をの件数や登録された件数を画面上に表示するかどうかを指定します。
names : PL/SQL を利用する場合の引数にセットすべき データの名称を指定します。
StopZero : 検索結果が0件のとき処理を続行するかどうかを指定します。
>
SQL 文
</mis:query>

2. update タグ

update タグは、データベース登録に使用されるタグです。
queryType の指定により、実際に呼び出す Query クラスを指定します。
通常は、queryType = “JDBCPLSQL” が良く使われますが、引数付き SQL 文を直接実行する場合は, ” JDBCPrepared” を指定します。

<mis:update
tableId : (通常は使いません)結果を DBTableModel に書き込んで、session に登録するときのキーを指定します。
queryType : 登録を実行する手段を指定します。
dbid : (通常は使いません)Query オブジェクトを作成する時の DB 接続 ID を指定します。
scope : キャッシュする場合のスコープを指定します。
command : 処理コマンドをセットします。
debug : デバッグ情報(タグのボディー部)を 出力する／しないを指定します。
displayMsg : 検索結果をの件数や登録された件数を画面上に表示するかどうかを指定します。
names : PL/SQL を利用する場合の引数にセットすべき データの名称を指定します。
>
SQL 文
</mis:update>

3. queryType

queryType 属性は、実際に起動される Query インターフェースの派生クラス名を指定します。
基本的に、query タグは、リクエスト情報から、update タグは、DBTableModel情報から、値を取り出して、queryType 属性で指定のクラスに引数を渡します。

queryType	説明	使用可能タグ
JDBCCallable	Query インターフェースを継承した 実装クラスです。java.sql.CallableStatement を用いて、データベース検索処理を行います。引数は、従来の PL/SQL の実行が可能なように、第一引数はエラーコード、第二引数は、エラーメッセージを返してきます。第三引数以降は、自由に指定できます。	query/ update
JDBCErrMsg	Query インターフェースを継承した 実装クラスです。java.sql.CallableStatement を用いて、データベース検索処理を行います。引数を配列指定で渡すことができ, エラー時には、DBErrMsg オブジェクトにエラー情報を格納して返すことが可能です。	query カーソルを返します。

queryType	説明	使用可能タグ
JDBCKeyEntry	EntryQuery タグで使用します。java.sql.CallableStatement を用いて、データベース検索処理を行います。引数に、キーと値をセットで配列指定で渡すことができ，エラー時には、DBErrMsg オブジェクトにエラー情報を格納して返すことが可能です。	entryQuery
JDBCPLSQL	Query インターフェースを継承した 実装クラスです。java.sql.CallableStatement を用いて、データベース検索処理を行います。引数に、SYSARG_ARRAY と、USERARG_ARRAY を配列指定で渡すことができ，エラー時には、DBErrMsg オブジェクトにエラー情報を格納して返すことが可能です。	update
JDBCPrepared	Query インターフェースを継承した 実装クラスです。java.sql.PreparedStatement を用いて、データベース検索処理を行います。引数に、指定した値を配列で渡します。	query/ update
JDBC	Query インターフェースを継承した 実装クラスです。java.sql.Statement を用いて、データベース検索処理を行います。引数は無しです。(与えられた SQL 文を実行します。)	query
JDBCUpdate	Query インターフェースを継承した 実装クラスです。java.sql.CallableStatement を用いて、データベース登録処理を行います。引数は、そのまま配列に格納して処理を行います。エラー時の処理や、検索結果の取り出しはできません。	query/ update

第14章 View 、WriteTable 、ReadTable

この章では、取り出したデータの出力方法について、説明します。通常は、標準出力（画面表示）ですが、ファイルへの入出力方法についても、説明します。

1. view タグ

view タグは、DBTableModel を表示するためのタグです。通常は，query タグにより、DBTable Modelが作成され、指定の scope に登録されるので、そこから取り出して、表示します。
表示する場合に、表示方法を、viewFormId 属性で指定された派生クラスを使用して表示します。

<mis:view

viewFormId	: session から取得する ViewForm オブジェクトの ID。
viewFormType	: ViewForm オブジェクトを作成する ViewFormFactory に与える ID。
scope	: キャッシュする場合のスコープを指定します。
command	: 処理コマンド
startNo	: 表示データを作成する場合の表示の開始行番号をセットします。
pageSize	: 表示データを作成する場合の1ページの行数をセットします。
columnWritable	: 書き込み可能カラム名を、カンマ区切りで与えます。
noWritable	: 書き込み不可カラム名を、カンマ区切りで与えます。
columnDisplay	: 表示可能カラム名を、カンマ区切りで与えます。
noDisplay	: 表示不可カラム名を、カンマ区切りで与えます。
language	: 言語コードを指定します。
tableId	: session から取得する DBTableModel オブジェクトの ID。
writable	: rowWritable（行が書き込み可能かどうか）を設定します。
checked	: 書き込み可能な行（rowWritable == true）のチェックボックスに対して初期値を 選択済みにするか、非選択済みにするかを指定します。
pagePlus	: 1ページの行数の増加分をセットします。
rowspan	: 表示データを作成する場合のフォーマットの行数をセットします。
skip	: 書き込み可能な行（rowWritable == true）のみを表示対象とするかどうかを指定します。
viewLinkId	: request から取得する ViewLink に対応する Attributes オブジェクトの ID。
selectedType	: 表示時の選択用オブジェクトのタイプを指定します。
optionTypeAttributes	: テーブル等のチェックボックスに属性を付加します。JavaScript などの HTML 基本タグ以外の属性を、そのままチェックボックス/ラジオボタン等に使用します。
noMessage	: 検索結果メッセージを表示する/しないを設定します。
backLinkCount	: ページの先頭へのリンクの間隔をセットします。

/>

Body 部分の存在する場合（主に、Format 系クラス）は、以下のようになります。

例:

```
<mis:view
  viewFormType = "HTMLCustomTable"
 ・・・・・・
>
  <thead> ・・・・・ </thead>
  <tbody> ・・・・・ </tbody>
  <tfoot> ・・・・・ </tfoot>
</mis:view>
```

viewFormType 属性

viewFormType 属性は、mis.pdm.hayabusa.html.XXXXViewForm.java クラスのXXXX部分を指定します。
これは、ViewForm インターフェースを実装した派生クラスです。
このクラスを切り替えることで、DBTableModel を色々な形で表示させることが可能になります。

viewFormType	説明
HTMLCalendar	1ヶ月分のカレンダー形式で、検索結果を表示するカレンダー表示クラスです。
HTMLCustomTable	ヘッダー情報、フッター情報、ボディー情報を指定することで、自由にテーブル表示のレイアウトが可能な、カスタムテーブル作成クラスです。
HTMLDynamic	DBTableModel から 各フィールド情報を動的にカラムを作成して表示する、動的カラム一覧表示クラスです。
HTMLFormatTable	ヘッダー部分のフォーマット情報に応じたテーブルを自動作成する、フォーマットテーブル作成クラスです。
HTMLFormatTextField	フォーマットを外部から指定することにより、自由にレイアウトを作成できる、テキストフィールド作成クラスです。
HTMLTable	DBTableModel から HTML 環境で取り扱うための,テーブルタグを自動生成する、テーブル作成クラスです。
HTMLTextField	DBTableModel から HTML 環境で取り扱うための,テキストフィールドタグを自動生成する、テキストフィールド作成クラスです。
HTMLTreeBOM	DBTableModel のレベルから、JavaScript のツリー階層を作成し、ツリー階層を持ったテーブル表示を行う、ツリーテーブル表示クラスです。

2. writeTable タグ

writeTable タグは、DBTableModel をファイルに書き出すためのタグです。このタグにより書き出された、DBTableModel は、readTable タグにより、もとの DBTableModel に戻すことが可能になります。query タグ、update タグ等と組み合わせて、データベース、ファイルをシームレスに連携させることが可能になります。

また、writerClass 属性に、DBTableWriter インターフェースの派生クラスを適用することで、各種形式(TSV 形式、CSV 形式、XML 形式、フラットファイル形式等)で、セーブできます。

```
<mis:writeTable
    separator       : 可変長ファイルを作成するときの項目区切り文字をセットします。
    HeaderSequence
                    : Label,Name,Size,Class,Data の各フィールドの頭文字のアルファ
                    ベットで出力順をセットします。
    fileURL         : fileURL を セットします。
    filename        : ファイルを作成するときのファイル名をセットします。
    encode          : ファイルを作成するときのファイルエンコーディング名をセットします。
    writerClass     : 実際に書き出すクラス名(の略称)をセットします。
    fileAppend      : 追加モードで書き込むかどうかをセットします。
    direct          : 結果をダイレクトに EXCEL ファイルとして出力するかどうかをセットします。
    zip             : 結果をファイルに出力するときに、ZIP で圧縮するかどうかをセットします。
    language        : 言語コードを指定します。
    tableId         : session から所得する DBTableModel オブジェクトの ID。
    scope           : キャッシュする場合のスコープを指定します。
/>
```

writerClass 属性を、以下に示します。

writerClass	説明
CSV	可変長カンマ区切り文字ファイルを出力します。
Default	可変長タブ区切り文字ファイルを出力します。
Fixed	固定長文字ファイルを出力します。
Properties	プロパティーファイルを出力します。
XML	XML 形式(ORACLE XDK 形式)ファイルを出力します。

コラム　ORACLE XDK 形式

writerClass の XML について、簡単に説明しておきます。
これは、ORACLE XDK(XML Development Kit) で使用できるXML形式でファイルをセーブすることで、XDKのユーティリティを用いてデータベースへ登録することが可能になります。
また、この形式は, XSQLサーブレットの出力形式でもあるため、各種文献やサンプル等で、XSLTファイルや、CSSファイルなどの流用が可能と思われます。

このXML は、下記のような形式をしています。
ROWSET と ROW があり、ROW の属性の num が行番号に相当します。
そして、各カラムがタグになり、データがBODY部に書かれています。

```
<?xml version = '1.0' encoding = 'Shift_JIS'?>
<ROWSET>
  <ROW num="1">
    <UNIQ>2</UNIQ>
    <SYSTEM_ID>DBDEF</SYSTEM_ID>
    ・・・・・・・
  </ROW>
  <ROW num="2">
    <UNIQ>1</UNIQ>
    <SYSTEM_ID>DBDEF</SYSTEM_ID>
    ・・・・・・・
  </ROW>
  <ROW num="3">
    <UNIQ>3</UNIQ>
    <SYSTEM_ID>DBDEF</SYSTEM_ID>
    ・・・・・・・
  </ROW>
  ・・・・・・
</ROWSET>
```

この形式のファイルをデータベースに登録するには、以下のコマンドが使えます。

```
java OracleXML putXML -user "HYBS/HYBS" -conn "jdbc:oracle:thin:@hn50g5:1521:HYBS"
-commitBatch 0 -fileName "%1.xml" "%1"
```

同様に、データベースから抜き出すには、以下のコマンドが使えます。

```
java OracleXML getXML -user "HYBS/HYBS" -conn "jdbc:oracle:thin:@hn50g5:1521:HYBS"
-encoding "Shift_JIS" "select * from %1"  >  %1.xml
```

もちろん、 Javaで開発ができるため、自分でこれらのAPIを使いながら、登録/更新などを行うクラスを独自に作成することも可能です。

参考:http://otn.oracle.co.jp/software/tech/xml/xdk/index.html

3. readTable タグ

readTable タグは、DBTableModel をファイルから読み出すためのタグです。通常は, writeTable タグにより、書き込まれた DBTableModel を再現するために使用されます。ファイルから読み出す場合に、読み出し方法を、readerClass 属性で指定された派生クラスを使用して表示します。

<mis:readTable
separator : 可変長ファイルを作成するときの項目区切り文字をセットします。
fileURL : fileURL を セットします。
filename : ファイルを作成するときのファイル名をセットします。
encode : ファイルを作成するときのファイルエンコーディング名をセットします。
readerClass : 実際に読み出すクラス名（の略称）をセットします。
maxRowCount : 読取時の最大取り込み件数をセットします。
language : 言語コードを指定します。
tableId : session から所得する DBTableModel オブジェクトの ID。
command : 処理コマンド
modifyType : ファイル取り込み時の モディファイタイプ（A,C,D 属性）を指定します。
/>

readerClass 属性は、DEFAULT のタブ区切りのみ用意しています。

第15章 高度なデータアクセス

この章では、通常のSQLによるデータ取り出しと、カラムオブジェクトの機能を組み合わせることによって、高速で汎用性の高いデータ取り出し方法について、説明します。

一般に、Web アプリケーションは、Web サーバーとデータベースサーバーという2階層か、それ以上の階層で構築されています。
この多階層化のメリットのうち、容易にスケーラビリティーを確保できるということがあります。
データベースサーバーのクラスタ化は、非常に高度な技術が要求され、かつ高価になる傾向があるため、Web サーバー側への負荷分散をまずは考慮すべきでしょう。

そうする場合に、Web サーバー側とデータベースサーバー側の負荷の振分においては、できるだけ Web サーバー側での処理を行い、データベースサーバー側は必要最小限、つまりデータの検索に特化するようにシステムを構築しておけば、負荷が増えてきた場合には、まず、Web サーバーの増設を行うだけで簡単に対応できます。

本フレームワークでは、検索結果のデータに対して、カラムオブジェクトを適用させることで各種チェックや表示形式の変換などを、データベースサーバーではなく、Web サーバーで実行しています。また、検索結果は、Web サーバーのDBTableModelにキャッシュし、管理しているため、次ページ操作などは、データベースにアクセスすることなく処理できます。同様に、検索結果に対する追加，修正，削除も、Web サーバー上で行った後、変更データ分のみをデータベースサーバーに登録するため、データベースサーバー側の負荷を低減できます。

ただし、Web サーバーには、十分なメモリを確保しておく必要があります。

次に、一般的な検索と表示の例を示します。

例)

```
<mis:query
        command         = “{@command}”
        maxRowCount     = “10000”
>
  <jsp:text>SELECT * FROM CUSTOMER</jsp:text>
</mis:query>

<mis:view
        viewFormType = ″HTMLTable″
        command         = ″{@command}″
        pageSize        = ″100”
/>
```

query タグの maxRowCount = “10000” は、データベースから 最大10000行を検索することを指定しています。これは、誤って大量のデータ検索を行った場合に、Webサーバー側のメモリ不足を引き起こさないための配慮です。
デフォルト設定は、SystemResource.properties の DB_MAX_ROW_COUNT パラメータで指定できます。

view タグの pageSize = ″100” は、画面上に表示する1ページ分の行数です。つまり、先に最大件数分を検索した場合（10000件検索）100件づつ画面に表示されます。このとき、画面を、NEXT, PREVした場合でも、データベースへの検索を行わずに、Webサーバー側のメモリ上に存在しているDBTableModelを再表示しているだけです。

また、画面に表示する場合に、数字であればカンマ編集や、日付であれば, YYYY/MM/DD の表示形式の変換は, データベースの検索結果にWebサーバーのカラムオブジェクトの表示用レンデラーを適用して処理しています。
例えば、10000件検索したとしても、画面上に100件検索しているだけですので、この100件分のみ、表示用レンデラーが適用されます。残りの9900件は、表示されるまで変換されません。

この特性を利用して、検索結果の表示処理（表示用レンデラー）で、特殊処理を行うことにより、パフォーマンスを上げることが可能になります。

以下に、その例となるレンデラーを取り上げます。

1. FORM レンデラー

FORM レンデラーは、パラメータで指定された FORM を表示するクラスです。
元の Value を、$1 として、使用することが可能です。
例）
DBColumnResource.properties に、FORMレンデラーの指定とパラメータに下記の様に指定する。

```
<a href=″../TEST14/index.jsp?          //1行で記述
        command=NEW&                   //1行で記述
        SYSTEM_ID=DBDEF&               //1行で記述
        TABLE_NAME=$1″                 //1行で記述
        target=″CONTENTS″ >$1</a>
```

これは、検索結果の値を、$1 の箇所に埋め込んで、リンク情報を作成します。
先に述べましたように、この変換処理は、画面に表示するときにのみ、行われます。

FORMレンデラーのパラメーターは何でも良く、image タグと組み合わせたリンクや定型の文書などでも構いません。

2. QUERY レンデラー

QUERY レンデラーは、パラメータで指定された SQL 文を実行し、その結果を表示するクラスです。
元の Value を、$1 として、使用することが可能です。
例)
DBColumnResource.properties に、QUERY レンデラーの指定とパラメータに下記の様に指定する。

```
Select count(*) from $1
```

これは、検索結果の値を、$1 の箇所に埋め込んで、テーブルのカウントを取ります。
先に述べましたように、この検索処理は、画面に表示するときにのみ、行われます。

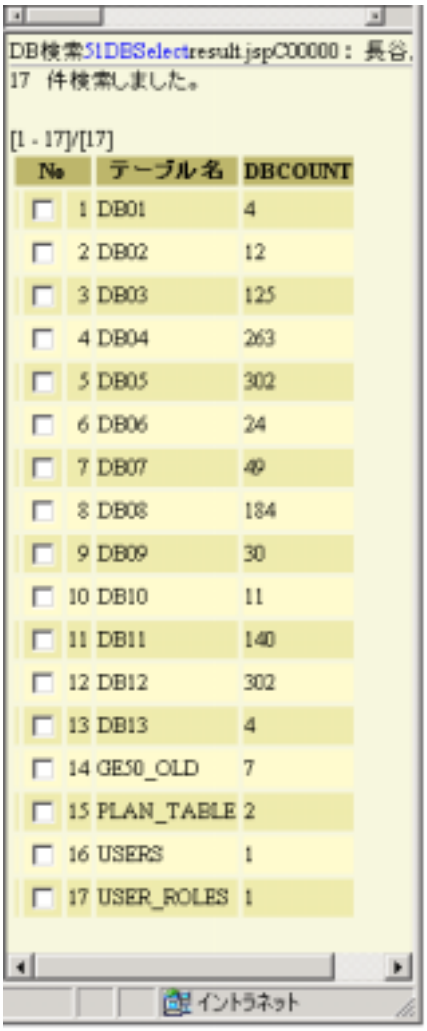
例)
DBColumnResource.properties に、DBCOUNT というカラムを定義する。

```
DBCOUNT=VARCHAR2 128 true QUERY TEXT XK _ "select count(*) from $1"
```

検索は、TABLE_NAME に別名として
DBCOUNT をつける。
元のテーブル名も必要なので、検索しておく。

```
<mis:query
  command="{@command}>
  <jsp:text>
        SELECT TABLE_NAME ,
        TABLE_NAME DBCOUNT
        FROM USER_TABLES
        ORDER BY TABLE_NAME
  </jsp:text>
</mis:query>

<mis:view
        viewFormType = "HTMLTable"
        command      = "{@command}"
/>
```



第 VI 部　ダイレクト・パス・インポートとダイレクト・パス・エクスポート

『うまく使えば、時間はいつも十分にある。』
ゲーテ

ここでは、ダイレクトパスについて説明します。
構成は、次のとおりです。

第 16 章　ダイレクト・パス・インポート
通常の DBTableModel に一旦キャッシュする方式では、巨大なデータをすべてメモリ中に管理することができないため、直接ファイルからデータベースに登録する方法について説明します。

第 17 章　ダイレクト・パス・エクスポート
同様に、直接データベースからファイルに抜き出す方法について説明します。

第16章 ダイレクト・パス・インポート

この章では、通常の DBTableModel に一旦キャッシュする方式では、巨大なデータをすべてメモリ中に管理することができないため、直接ファイルからデータベースに登録する方法について説明します。

【現在調整中です。】

第17章 ダイレクト・パス・エクスポート

この章では、同様に、直接データベースからファイルに抜き出す方法について説明します。

【現在調整中です。】

第 VII 部　アプリケーションの保護

『弓の弦は二本持っていなければならぬ。』
シャルル・ド・ボヴェル

ここでは、アプリケーションの保護について説明します。
構成は、次のとおりです。

第 18 章　アプリケーション・アクセスの制御
アプリケーション・リソースへのユーザー・アクセスを制御する方法について説明します。

第 19 章　権限、ロールおよびセキュリティ・ポリシー
ユーザーの設定と画面の設定方法について説明します。

第 20 章　アプリケーション監査
使用メモリ、ログインユーザーの監視、および、キャッシュクリア、コネクションの破棄等管理画面の操作を中心に説明します。

第18章 アプリケーション・アクセスの制御

この章では、アプリケーション・リソースへのユーザー・アクセスを制御する方法について説明します。

アプリケーションアクセスに関する制御には、IPアドレスによる制限や、ドメインユーザーなどの認証に関する制限などの、半固定にちかい制限と、アプリケーションレベルの流動性の高い、いわば、その都度変更できる制限があります。

ここでは、アプリケーションレベルでの制限について，述べます。

Webアプリケーションフレームワークでは、すべてのログインユーザーを管理する必要があります。それは、例え，ユーザーが自らログインせずに、GUESTユーザーとして自動的に扱われたとしても、同様です。
ユーザー情報は、セッションごとに作成されます。よって、同一ユーザー（例えば、GUEST）が同時に複数ログインしたとしても、セッションが別であれば、別人として扱われます。
（ただし、GUESTユーザーの属性は同一です。また、テンポラリファイル等を使用する場合は、一般的なアプリケーションの作りをしている場合は、同一になります。よって、そのような使い方・・・・つまり、書込みを行う場合は、きちんとログインユーザーを分けるべきです。）

本フレームワークでは，ユーザー認証と、画面等のアクセス制限を分けて管理しています。
ユーザー認証は、Tomcat等のサーブレットコンテナに任せて、アクセス制限を、ユーザーリソースとGUIリソースを用いて処理しています。

1. ユーザー認証

ユーザー認証は、Tomcat等のサーブレットコンテナで行っています。
以下、Tomcatを例に説明します。
設定ファイルは、 APPS¥java¥tomcat¥conf¥ server.xml の Realm で設定します。
デフォルトの MemoryRealm を使用するのではなく、JDBCRealm を用いてデータベース上のユーザーで認証を行います。
このテーブルと、後ほど説明するアクセス制限用のユーザーテーブルを同一にすることで、メンテナンス工数の削減が可能になります。

```
<Realm   className="org.apache.catalina.realm.JDBCRealm" debug="99"
         driverName="oracle.jdbc.driver.OracleDriver"
         connectionName="HYBS"
         connectionPassword="HYBS"
         connectionURL="jdbc:oracle:thin:@hn50g5:1521:HYBS"
         userTable="GE10" userNameCol="USERID" userCredCol="PASSWD"
         userRoleTable="GE10" roleNameCol="SYSTEM_ID" />
```

2. セキュリティ制限

セキュリティ制限は、JSP／Servlet準拠の web.xml で対応しています。
先のユーザー認証は、サーブレットコンテナ（例 Tomcat）ごとに異なりますが、web.xml はJSPの規格に準拠していますので、共通に使用できます。

UAP¥webapps¥システムID¥WEB-INF¥web.xml より抜粋

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected JSP Area</web-resource-name>
    <url-pattern>/jsp/*</url-pattern>
    <url-pattern>/help/*</url-pattern>
    <url-pattern>/filetemp/*</url-pattern>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>*</role-name>
  </auth-constraint>
</security-constraint>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected Other Area</web-resource-name>
    <url-pattern>/src/*</url-pattern>
    <url-pattern>/log/*</url-pattern>
    <url-pattern>/def/*</url-pattern>
    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>XXXXXXXXXXXXXXXX</role-name>
  </auth-constraint>
</security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Web App</realm-name>
  </login-config>
```

3. directory listings

Webサーバーで、あるディレクトリをアクセスしたときに、そのディレクトリ一覧を表示させるのか、または、表示させないのかを指定するには、default servlet の設定を、行う必要があります。
listings 属性を false に設定すると、ディレクトリの表示を行わないようになります。
なお、ディレクトリ表示を行う場合、welcome-file の設定があると、そのファイルを表示してしまいますので、ご注意ください。

```
APPS¥java¥tomcat¥conf¥web.xml
  <!--  listings      Should directory listings be produced if there -->
  <!--                is no welcome file in this directory?  [true]  -->
<servlet>
<servlet-name>default</servlet-name>
<servlet-class>
org.apache.catalina.servlets.DefaultServlet
</servlet-class>
<init-param>
<param-name>debug</param-name>
<param-value>0</param-value>
</init-param>
<init-param>
<param-name>listings</param-name>
<param-value>false</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

4. Digest認証

先の認証の例は、BASIC認証といわれる方式で、ユーザー名とパスワードをBase64形式でエンコードして送信しますので、転送中に傍受されると容易に解読されてしまいます。
DIGEST 認証は、無作ために生成した文字列(ナンス)をクライアントに返し、それを用いてURL, HTTP方式、ナンスから一方向ハッシュ(MD－5)を作成し、サーバーは、独自のチェックサムで比較して判定します。

例:BASIC認証時のヘッダー情報から、ユーザー/パスワードの取得方法

```
String authorization = request.getHeader("Authorization");
String userInfo = authorization.substring( 6 ).trim();
BASE64Decoderde coder = new.BASE64Decoder();
String nameAndPasswd = new String( decoder.decodeBuffer( userInfo ) );
int index = nameAndPasswd.indexOf( ":" );
String user     = nameAndPasswd.substring( 0,index );
String passwd = nameAndPasswd.substring( index+1 );
```

Digest認証の設定は、web.xml で記述します。

UAP¥webapps¥システムID¥WEB-INF¥web.xml より抜粋

```
<login-config>
  <auth-method>DIGEST</auth-method>
  <realm-name>Web App</realm-name>
</login-config>
```

5. HTTPSクライアント認証

SSL(Secure Sockets Layer)は、公開キーと対象キー暗号を組み合わせてセキュアセッションを確立します。

APPS¥java¥tomcat¥conf¥ server.xml の下記のコメントアウトされている箇所を復帰します。

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
           port="8443" minProcessors="5" maxProcessors="75"
          enableLookups="false"
           acceptCount="10" debug="0" scheme="https" secure="true">
    <Factory className="org.apache.catalina.net.SSLServerSocketFactory"
             clientAuth="false" protocol="TLS"/>
</Connector>
```

さらに UAP¥webapps¥システムID¥WEB-INF¥web.xml の security-constraint に、user-data-constraint を追加します。

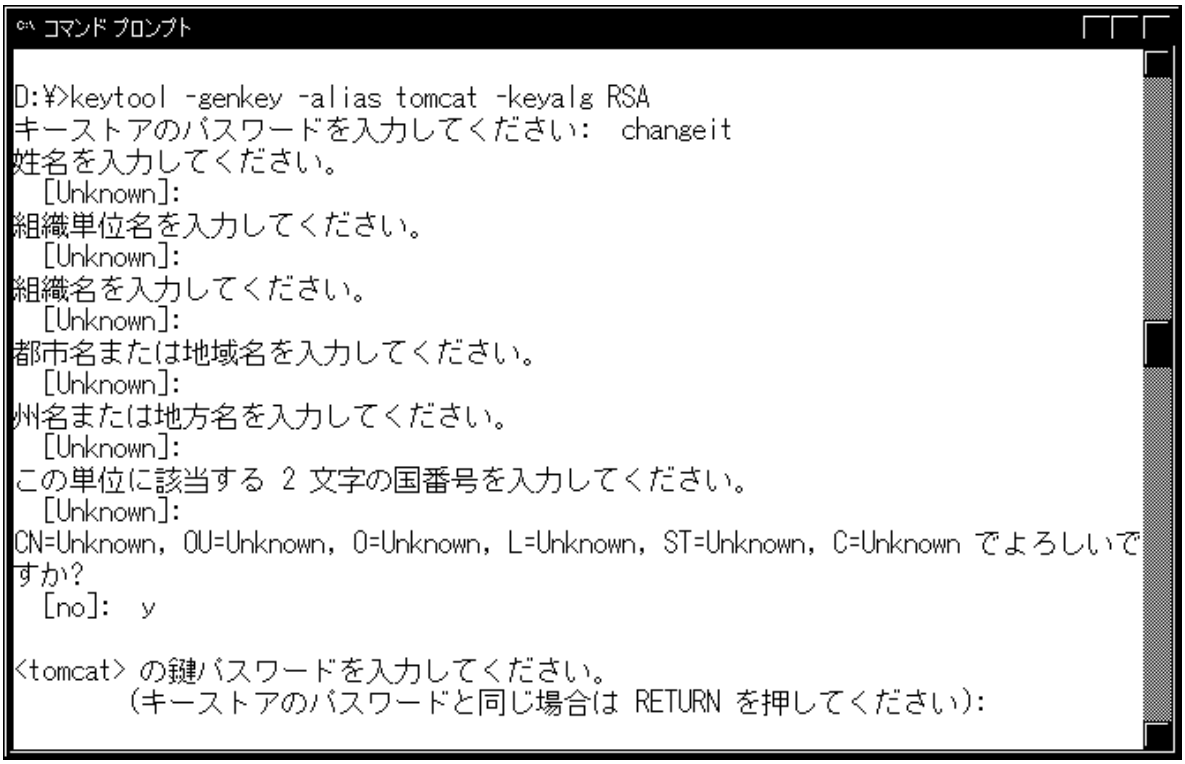
```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Protected JSP Area</web-resource-name>
    <url-pattern>/jsp/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>*</role-name>
  </auth-constraint>
  <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

SSLは、JDK1.4 より標準パッケージになりましたので、そのまま使用できます。
それ以前のJDKでは、JavaTM Secure Socket Extension (JSSE) をインストールする必要があります。
参考:http://java.sun.com/products/jsse/

注意：Webエンジン Ver 3.0 では、JDK1.4 以上となっています。

エイリアス『tomcat』に公開キーと秘密キーを生成する必要があります。これは、JDKの keytool を使用して，作成してください。

keytool -genkey -alias tomcat -keyalg RSA



第19章 権限、ロールおよびセキュリティ・ポリシー

この章では、ユーザーの設定と画面の設定方法について説明します。

1. 画面メニューのアクセス制限

画面メニューのアクセス制限は、ユーザーリソースとGUIリソースの設定により、個々の画面に対して、読取許可、書込み許可を与えることが可能です。

ユーザーリソースのロール、グループ、プロジェクトと、GUIリソースのロール、グループ、プロジェクト、アクセスモードで判断されます。

以下に、アクセスチェックの手順を示します。

1. ユーザー情報のロールが、"root" であれば、アクセスモードの rw 属性のみで判定
2. アクセスモードのその他(rwrwrwrw の最後の rw)のチェックで判定
3. 以下の判定は、全てが成立している必要がある。
   (ア) ロール、グループ、プロジェクトのロールのどれかに r 属性または w 属性のいずれかが設定されている。
   (イ) 画面ロールがあり、かつ、ユーザーロールを含む。
   (ウ) 画面グループがあり、かつ、ユーザーグループを含む。
   (エ) 画面プロジェクトがあり、かつ、ユーザープロジェクトを含む。

上記判定で、アクセス許可が与えられます。

2. 画面の登録ボタンのアクセス制限

アクセスモードの rw 属性は、読取許可(r)と書込許可(w)で指定します。
制限をかける場合は、- を記入してください。

例)	rwrwrwrw	フルアクセス(その他モードも rw のため)
	rwrwrw--	指定のユーザーのみアクセス可能
	r-r-r-r-	すべてで、読取専用
	-w-w-w-w	書き込み専用・・・メニューには現れません。

個々の画面のロール、グループ、プロジェクトと、ユーザーのロール、グループ、プロジェクトを制御することで、さらに画面単位にユーザーごとのアクセス制限をかけられます。

また、読取権限のみに制限されている場合は、登録ボタンが表示されません。
登録ボタンの制限は、JSP画面の、writeCheck タグで行っています。

例） writeCheck による書込制限判定

```
<!-- 追加、複写、変更、削除ボタンを作成します。 -->
<mis:writeCheck>
    <mis:input type="submit" name="command" value="COPY"
                msg="MSG0035" accesskey="C" td="false" />
    <mis:input type="submit" name="command" value="MODIFY"
                msg="MSG0036" accesskey="M" td="false" />
    <mis:input type="submit" name="command" value="DELETE"
                msg="MSG0037" accesskey="D" td="false" />
    <jsp:directive.include file="../common/Excel_direct.jsp" />
</mis:writeCheck>
```

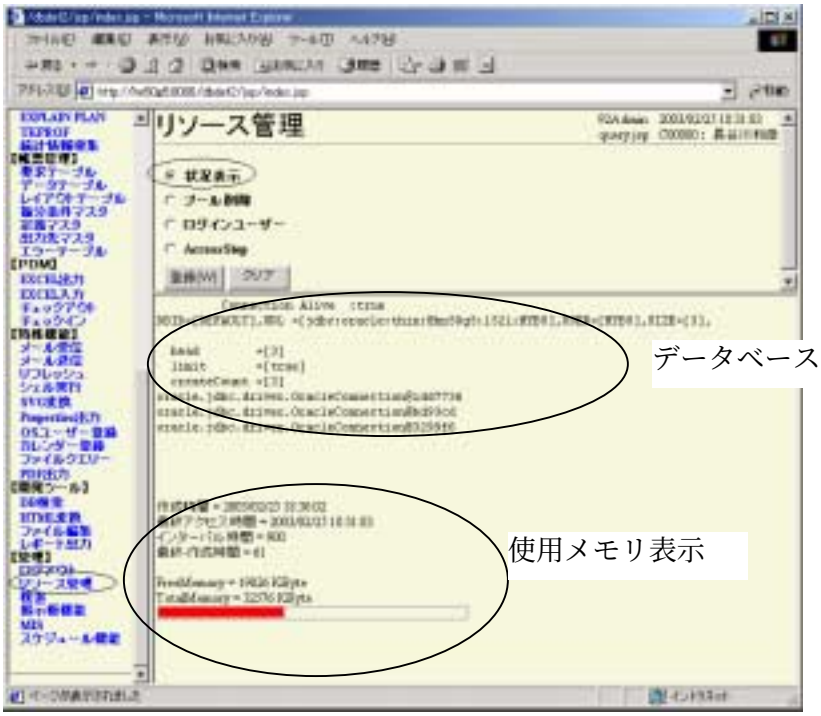
このタグで囲われた範囲は、書込み許可が与えられない場合は、表示されません。

第20章 アプリケーション監査

この章では、使用メモリ、ログインユーザーの監視、および、キャッシュクリア、コネクションの破棄等管理画面の操作を中心に説明します。

1. 使用メモリ、コネクション情報

使用メモリ、および、使用データベースコネクションについての情報は、リソース管理メニューの状況表示より検索できます。



使用メモリの算出方法は、Runtime オブジェクトより取得します。

```
int freeMemory    = (int)( Runtime.getRuntime().freeMemory()/1024 ) ;
int totalMemory = (int)( Runtime.getRuntime().totalMemory()/1024 );
int useMemoryRatio = (int)(((totalMemory − freeMemory) * 100 )/totalMemory) ;
```

2. ログインユーザーの監視

現在ログインしているユーザーの一覧を表示します。
ログインしている状態とは、セッションが残っている状態のセッション数を指します。
つまり、同一人物が複数セッションでログインしていても、セッション分（複数）接続とみなします。

ログインユーザーの一覧を
表示します。

3. キャッシュクリア、コネクションの破棄

内部でキャッシュしているリソース情報や、データベース接続キャッシュを破棄します。
データベースコネクション上でエラーが発生した場合は、自動的に破棄しますが，]想定ケース以外の状況で、不正なセッションが残ったままの場合は、エラーになります。その場合は、セッション破棄で、クリアする必要があります。

索引