Webアプリケーション管理者ガイド

Webエンジン Ver. 3.5対応
リリース 1.0
初版:2003 年 10 月

Muratec Information Systems.LTD.

Ｗｅｂアプリケーション管理者ガイド、 リリース 1.0
原本部品番号:W2A0001-01
原 本 名: Hayabusa Web Application Administrator's Guide, 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、システム管理者がWebアプリケーションを使用した環境で、システム管理をする際に必要となる知識を習得します。インストール、セキュリティ、チューニングといった知識・技術を習得します。このようなユーザは、Webアプリケーションの作成、円滑な運用、使用状況の監視などの役割を持ちます。

≪対象読者≫

Webアプリケーションの運用を管理するユーザを対象としています。
このマニュアルの読者は、MIS 認定 Web アプリケーション講習会入門コースを受講された方、あるいはWebアプリケーションの概念、Windows、Java、Tomcat等の基礎内容をよく理解しているものと想定しています。また、Webアプリケーションを稼働しているオペレーティング・システム環境もよく理解している必要があります。

≪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　インストールと環境設定

『何でも思い切ってやってみることですよ。
どっちに転んだって人間、
野辺の石ころ同様骨となって
一生を終えるのだから。』

坂本 竜馬

ここでは、Webアプリケーションのインストール方法と環境設定について説明します。
この部では、WebアプリケーションインストールCD－ROMが必要です。

構成は次のとおりです。

第 1 章　データベース設定
データベース設定の方法について説明します。

第 2 章　インストールと環境設定
Webアプリケーションのインストール手順、環境設定の方法について説明します。

第 3 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1章 データベース設定

この章では、データベース設定の方法について説明します。

1. 環境設定

CD-ROM より、UAP¥GE フォルダを、コピーします。
ここに、データベース作成用のスクリプトが入っています。

2. 環境構築スクリプト

UAP¥GE¥DEF¥DBDEF¥環境作成.sql が、環境構築スクリプトです。
このスクリプトは、ハンドで流すことを前提としています。

```
DEFINE SetupDB=G:\GE\DEF\DBDEF\SetupDB
DEFINE DBS=G:\GE\DBS
DEFINE TABLE_SET=&SetupDB\TABLE_SET

REM テーブルスペース １つ当りの大きさ (MB)
DEFINE TBLMB=10M
DEFINE IDXMB=10M

connect system/system
rem システム領域設定
rem @&SetupDB\SYSTEM_SET\AllSystemSetting.sql

@&SetupDB\SYSTEM_SET\CreateTableSpace.sql
@&SetupDB\SYSTEM_SET\CreateUser.sql

connect GE/GE
rem ユーザーテーブル設定
@&SetupDB\TABLE_SET\AllTableSetting.sql
```

赤字(下線)の個所が、変更の必要なところです。
DBDEF には、リソース DB 以外のテーブルも存在します。

この、スクリプトにより、以下の場所にテーブルスペースが作成されます。

データ領域：

GETBL01	１０M	G:\GE\DBS\DATA\GETBL01.dbf
GETBL02	１０M	G:\GE\DBS\DATA\GETBL02.dbf

インデックス領域：

GEIDX01	１０M	G:\GE\DBS\INDEX\GEIDX01.dbf
GEIDX02	１０M	G:\GE\DBS\INDEX\GEIDX02.dbf

第2章 インストールと環境設定

この章では、Webアプリケーションのインストール手順、環境設定の方法について説明します。

1. Webアプリケーション設定

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:
・初期使用メモリ　例) -Xms128m
・最大使用メモリ　例) -Xmx128m
・認証用接続DB情報　例) jdbc:oracle:thin:@hn51d4:1521:HYBS
・接続先DB情報　例) jdbc:oracle:thin:@hn51d3:1521:ORCL

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:）と設定しています。

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。
CD-ROM¥APPS¥java¥jdk14
CD-ROM¥APPS¥java¥tomcat4.1.27
※下線部分は、各アプリケーションのバージョンが入ります。

通常 JDK に関しては、各自でインストールしてください。
ただし、デフォルトでは、JRE や PATH が自動的に設定されてしまうため、③以下の設定と競合する可能性があります。できれば、JDK をインストール後、パス等は、init.bat （後述）を利用して起動ごとに設定するようにしてください。

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。
CD-ROM¥bin ,
CD-ROM¥webapps

webapps 以下に、各自の Web アプリケーションを配備します。デフォルトの、tomcat¥webapps は、使用しません。

④bin¥init.bat の書き換え（ドライブ設定、メモリ設定）

$UAP¥bin¥init.bat を書き換えます。

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

set CATALINA_OPTS=-server -Xms128m -Xmx128m   -Xss256k
                                  ↑   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

ヒント
設定メモリは、OS の空きメモリの約半分を目安にすればよいでしょう。 初期使用メモリと最大使用メモリを同じにすれば、メモリ拡張時の不要な処理を省けるため、若干高速化します。

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

$APPS¥java¥tomcat4.1.27¥conf¥server.xml を書き換えます。

- PORT設定、DNS逆引き停止
  当フレームワークでは、デフォルトの8080ではなく、8823を推奨しています。これは、他のサーバーのデフォルトポートとして、8080は比較的メジャーなため、ポート競合などの余計なトラブルを未然に防ぐためです。8823（ハヤブサ）と、覚えてください。
  DNS 逆引きは、クライアントからのアクセスログの見易さに関係しますが、一般に逆引きしない設定でも問題ありません。これは、パフォーマンスに影響します。

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
        port="8823" minProcessors="5" maxProcessors="75"
        enableLookups="false" redirectPort="8443"
        acceptCount="10" debug="0" connectionTimeout="60000"/>
```

- ドライブ設定
  デフォルトの tomcat¥webapps ではなく、先にインストールした UAP(G:ドライブ）に、デフォルト設定します。もちろん、Context で、個別に設定可能ですが、一ヵ所にまとめておいた方が、管理がしやすくなります。

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

- 認証DB設定
  エンジンでは、認証を標準で行う前提で、アプリケーションが組んであります。デフォルトの MemoryRealm（tomcat 4.1 では、UserDatabase ）による tomcat-users.xml ファイルでの認証では、やはり運用が難しいため、データベースを用いた認証を行ってください。

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

- コンテキスト設定
  個々の Web アプリケーションの設定を行います。

```
<Context path="/training" docBase="training" debug="0"
                 reloadable="true" crossContext="true">
          <Logger className="org.apache.catalina.logger.FileLogger"
                 prefix="localhost_training_log" suffix=".txt"
                 timestamp="true"/>
```

⑥SystemResource.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:@hn51d3:1521:ORCL

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。

ヒント
エンジン Ver3.5 より、リソース DB での運用を前提としています。リソースファイルでは、変更時に都度コンパイルが必要ですが、リソースDB では、設定時に、キャッシュをクリアするだけで連続的に使用可能です。

⑧bin¥startup.bat のスタートアップメニューへの登録

TOMCAT は、Windows のサービス化をしていません（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。
また、Windows 2000 のスタートアップメニューは、『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。

⑨Tomcat 実行

$UAP¥bin¥startup.bat をダブルクリックすれば、TOMCAT が起動し、$UAP¥bin¥shutdown.bat で停止しますので、動作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。

ヒント
起動と停止については、『第 3 章 Web アプリケーションの起動と停止』を参照してください。

⑩ブラウザでの動作確認

http://サーバー名:8823/プロジェクトID/jsp/index.jsp にアクセスして、正常に起動することを確認します。

第3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。ここでは、Tomcat サーバーを使用する場合を例にあげて、解説します。

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 -Xms128m -Xmx128m -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 II 部　リソース管理

『敵将は皆経験に富み、決して凡庸ではないが、
彼らは一時に多くのことを考えすぎた。
私は常に敵の主力のことだけを考えた。』

ナポレオン・ボナパルト

ここでは、リソースとその管理ツールについて説明します。
構成は次のとおりです。

第 4 章　リソース情報
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。

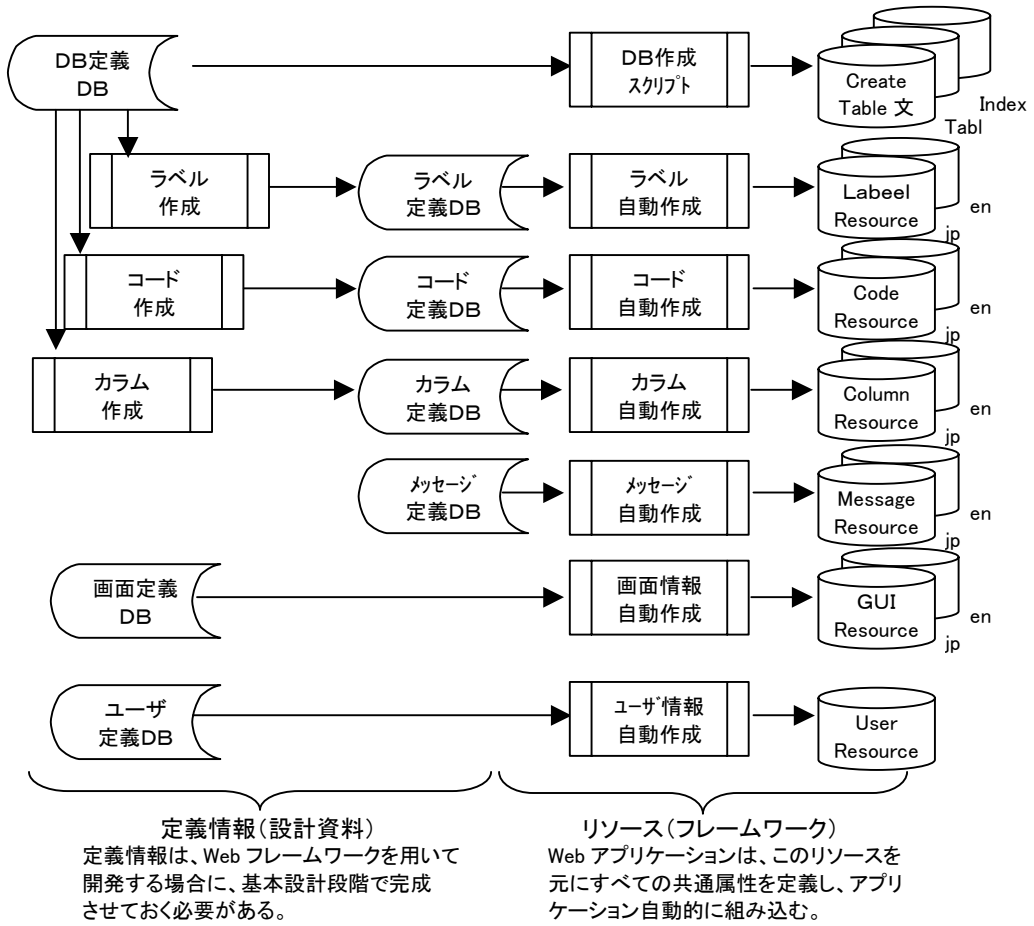
第 5 章　リソース管理ツール
リソースファイルを管理するための Web アプリケーションについて説明します。

第4章 リソース情報

この章では、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。

1. リソース管理

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です。当フレームワークでは、それらを一元管理し、ノン・コーディングで利用しています。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。
Ver3.5 より、リソース DB を推奨しますが、リソース DB から、リソースファイルへの変換が可能なので、開発時はリソース DB、実行時はリソースファイルという切り替えも可能です。



2. DB定義

DB 定義はリソースファイルの要の情報を作成する元情報となります。
DB定義書の要素は次のとおりです。

項目名	画面の表示ラベルとなる名称
カラム名	DB 定義のカラムとなり、Web アプリケーションで使用するキー情報。
物理バイト数	データベースチェック時や固定長抜き出し時に利用するカラムの文字数制限値
文字種別	通常の文字型(X)、数字型(S9)以外に、小文字(XL)、大文字(XU)、全角(K)、日付文字(YMD)、金額(YEN)などを定義し、この定義ごとに、DB 登録時に物理特性チェックを行う。
GUI種別	画面表示における表示形式を指定。表示形式と編集形式の両方のデフォルト設定である(例えば、通常のテキストフィールドやプルダウンメニューなど)。
ホスト送信	外部システムへの接続において、データを抜出す時に指定の桁数で抜き出せるように文字数を指定できる(固定長で抜き出す場合)。
初期値	指定がない場合の登録データ
備考	カラムの説明。ただし、プルダウンメニューの場合は、そのメニュー内容を A:内容 の形式で表し、コードリソースとして使用する。

項目ごとの論理チェック(マスタ DB の存在チェックなど)は、各システムごとに運用が異なるため、外部ファイルにてDBテーブルごとにチェック基準を設定し、そのチェックファイルを読み込んで処理します。これら以外に、データ登録時の制限や使用者ごとの制限(管理職は登録可能など)も考えられますが、現段階では画面のアクセス制限を利用してユーザーごとの制限を行います。

DB定義書サンプル

SEQ	ﾃｰﾌﾞﾙID	テーブル名	項番	項目名	カラム名	属性	使用桁数	物理桁数		インデックス					項目条件		ホスト送信	書込許可	表示種別	編集種別	文字種別	ﾃﾞﾌｫﾙﾄ	備考
SEQ	DBID	TBLNAME	NO	NAME	CLM	TYPE	LENU	LENP	UK	IA	IB	IC	ID	IE	NN	GE	LENH						BIKO
1	DB01	テーブルスペース定義	1	ユニークキー	UNIQ	NUMBER	9	22	#						○		-1	True	La	Text	S9		そのカラムを決定できるユニークな番号
2	DB01	テーブルスペース定義	2	プロジェクト	PROJECT	VARCHAR2	10	10		1					○		-1	True	La	Text	XU		
3	DB01	テーブルスペース定義	3	テーブル種別	TBLSHU	VARCHAR2	1	1							○		-1	True	La	Menu	XU		0:ネ
4	DB01	テーブルスペース定義	4	テーブルスペース名	TABLESPACE NAME	VARCHAR2	30	30		2					○		-1	True	La	Text	XU		
5	DB01	テーブルスペース定義	5	名称(漢字)	JNAME	VARCHAR2	50	50									-1	True	La	Text	XU		
6	DB01	テーブルスペース定義	6	追加フラグ	FGADD	VARCHAR2	1	1							○		-1	True	La	Menu	XU		0:初期登録 1:追加登録
7	DB01	テーブルスペース定義	7	データファイル	DATA FILE	VARCHAR2	128	128		3					○		-1	True	La	Text	XU		
8	DB01	テーブルスペース定義	8	自動計算データサイズ(MB)	SIZE AUTO	NUMBER		22									-1	True	La	Text	S9		テーブル定義より自動計算
9	DB01	テーブルスペース定義	9	データサイズ(MB)	DATA SIZE	NUMBER		22							○		-1	True	La	Text	S9	5	デフォルト値 = 5MB
10	DB01	テーブルスペース定義	10	自動計算繰	INITIAL AUTO	NUMBER		22									-1	True	La	Text	S9		テーブル定義より自動計算
11	DB01	テーブルスペース定義	11	初期エクステント(KB)	INITIAL EXTENT	NUMBER		22							○		-1	True	La	Text	S9	128	デフォルト値 = 128K
12	DB01	テーブルスペース定義	12	自動計算NEXTエクステント(KB)	NEXT AUTO	NUMBER		22									-1	True	La	Text	S9		テーブル定義より自動計算
13	DB01	テーブルスペース定義	13	NEXTエクステント(KB)	NEXT EXTENT	NUMBER		22							○		-1	True	La	Text	S9	128	デフォルト値 = 128K
14	DB01	テーブルスペース定義	14	最小エクステント	MIN EXTENTS	NUMBER		22							○		-1	True	La	Text	S9	1	デフォルト値 = 1
15	DB01	テーブルスペース定義	15	最大エクステント	MAX EXTENTS	NUMBER		22							○		-1	True	La	Text	S9	255	デフォルト値 = 255
16	DB01	テーブルスペース定義	16	エクステント増加率(%)	PCT INCREASE	NUMBER		22							○		-1	True	La	Text	S9	0	デフォルト値 = 0
17	DB01	テーブルスペース定義	17	ステータス	STATUS	VARCHAR2	1	1							○		-1	True	La	Menu	XU		0:オフライン 1:オンライン
18	DB01	テーブルスペース定義	18	コメント	COMMENTS	VARCHAR2	128	128									-1	True	La	Text	XU		
19	DB01	テーブルスペース定義	101	状態フラグ	FGJ	VARCHAR2	1	1									-1	True	La	Text	XU		
20	DB01	テーブルスペース定義	102	改廃コード	CDKH	VARCHAR2	1	1									-1	True	La	Text	XU		

３．システムリソース

システムリソースファイルは、他のリソースと異なり、データベース化されていません。このリソースを読み取って、各リソースをファイルからか、データベースからか判断するためです。つまり、システムを動作させる/システムの振る舞いを規定するために必要なリソースといえます。

主要な項目とその概要を以下に説明します。

```
#=========================================================================#
# これは，本システム全般に渡って,使用されるリソースです。                 #
# システムとしての初期値や，設定値などは，すべてここで登録されます。     #
# このリソースを修正する場合は,注意して下さい。                           #
# 不要な個所（値）を記入したり，記入漏れが発生すると,システムが           #
# 正常に起動しない場合があり得ますので,ご注意願います。                   #
#=========================================================================#

# ファイル出力 基準ＵＲＬ
FILE_URL = filetemp/

# ヘルプファイル 基準ＵＲＬ
HELP_URL = help/

# 帳票ファイル出力 基準ＵＲＬ
# 設定されていない場合は、FILE_URL + /REPORT/ に設定されます。
REPORT_FILE_URL = filetemp/REPORT/

# 帳票エンコーディングを設定します。
# 設定されていない場合は、UTF-8 に設定されます。
# この値は、固定値ですので、変更しないでください。
#
REPORT_ENCODE = UTF-8

#=========================================================================#
# これは，データベース接続先情報に関するリソースです。                   #
# ドライバー、ＵＲＬ，ユーザー、パスワードのほかに、キャッシュする       #
# コネクション数、最大同時接続数、コネクション取得待機時間等を設定       #
# します。                                                                #
#=========================================================================#

# データベースエンコーディングを設定します。
# オラクルのエンコーディング名ではなく、Java のエンコーディング名で指定します。
# Java とオラクル間の文字コード変換は、JDBC が自動で行います。
# ここでの設定は、データベース登録時の文字バイト数チェックに利用しています。
# DB_ENCODE = Shift_JIS
# DB_ENCODE = MS932
# DB_ENCODE = Windows-31J
```

```
DB_ENCODE = UTF-8

# ConnectionFactory.java
# 接続データベースのドライバー
# 複数登録する場合は,キー "DB_DRIVER" だけ共通で後ろを替えます。
DB_DRIVER_1 = oracle.jdbc.driver.OracleDriver
# DB_DRIVER_2 = mis.pdm.hayabusa.sql.MISDriver
# DB_DRIVER_3 = com.microsoft.jdbc.sqlserver.SQLServerDriver

# ConnectionFactory.java
# 接続データベースの設定  DEFAULT dbid
# DBID キーワード + 各設定キーワードで検索します。
#
# 設定キーワード は、URL,USER,PASSWD,最大プール数、初期接続数 です。
#
# DB_MAXCOUNT 最大プール数 は、コネクションプールの最大数です。
# この最大数が、同時接続がこの数を超えると、MissingResourceException が、
# 投げられますので、後程再接続する必要があります。
#
# DB_MINCOUNT コネクションプールの最小数 は、予めコネクションを準備しておきます。

#DEFAULT_DB_URL  = jdbc.mis:test:@pc4839:1521:DBRK

#DEFAULT_DB_URL  = jdbc:oracle:oci:@HYBS
DEFAULT_DB_URL  = jdbc:oracle:thin:@localhost:1521:HYBS
DEFAULT_DB_USER  = GE
DEFAULT_DB_PASSWD  = GE
DEFAULT_DB_MINCOUNT  = 3
DEFAULT_DB_MAXCOUNT  = 10


#=========================================================================#
# JSP で使用している変数
#=========================================================================#

# セッションタイムアウト（秒）
MAX_INACTIVE_INTERVAL = 900


#=========================================================================#
# mis.pdm.hayabusa.io.* で使用している変数
#=========================================================================#

# ファイルエンコーディングを設定します。
# FILE_ENCODE = Shift_JIS
```

```
# FILE_ENCODE = MS932
# FILE_ENCODE = Windows-31J
# FILE_ENCODE = UTF-8
# FILE_ENCODE = ISO-8859-1
# FILE_ENCODE = GB2312
# FILE_ENCODE = GB18030
FILE_ENCODE = UnicodeLittle

#=====================================================================#
# mis.pdm.hayabusa.db.* で使用している変数
#=====================================================================#

# ConnectionFactory.java
# コネクションを破棄するのに,何回リトライするか
DB_CLOSE_RETRY_COUNT = 10

# ConnectionFactory.java
# コネクションを破棄するリトライに, 何 MS の間隔でリトライを行うか
DB_CLOSE_RETRY_TIME = 1000

# JDBCTableModel.java
# コネクションを取得するのに,何回リトライするか
DB_RETRY_COUNT = 10

# JDBCTableModel.java
# コネクションを取得するリトライに, 何 MS の間隔でリトライを行うか
DB_RETRY_TIME = 1000

# JDBCTableModel.java
# データ検索時の最大件数
# この件数以上のデータは、物理的に取得できなくなります。
DB_MAX_ROW_COUNT = 1000

# AbstractQuery.java
# データ検索時の最大処理制限時間
# この時間（秒数）以上 SQL が終了しない場合は, MISServiceException を発行します。
DB_MAX_QUERY_TIMEOUT = 180

# ConnectionFactory.java
# Connection をプールしておく最大時間（秒）
# この時間（秒数）以上プールされている Connection は物理的に接続を終了させて
# 新しく Connection を作成します。
DB_MAX_CONNECTION_POOL_TIME = 900
```

```
#=====================================================================#
# mis.pdm.hayabusa.resource.* で使用している変数
#=====================================================================#

# リソースデータベースのシステム ID 名
SYSTEM_ID = GE

# リソースのプロパティーファイル名
# ラベルリソース
# RESOURCE_LABEL = resource.LabelResource
RESOURCE_LABEL_DB = SELECT CLM,LABEL_NAME FROM GE08 WHERE SYSTEM_ID in ( ?,'GE' )
AND LANG = ? ORDER BY CLM,KBSAKU
# RESOURCE_LABEL_DBID = GE

# コードリソース
# RESOURCE_CODE = resource.CodeResource
RESOURCE_CODE_DB = SELECT CLM,CODE(CLM,SYSTEM_ID,LANG) FROM GE04 WHERE SYSTEM_ID
in ( ?,'GE' ) AND LANG = ? GROUP BY CLM,CODE(CLM,SYSTEM_ID,LANG)
# RESOURCE_CODE_DBID = GE

# メッセージリソース
# RESOURCE_MESSAGE = resource.MessageResource
RESOURCE_MESSAGE_DB = SELECT MSGCD,MSGTXT FROM GE09 WHERE SYSTEM_ID in ( ?,'GE' )
AND LANG = ? ORDER BY MSGCD,KBSAKU
# RESOURCE_MESSAGE_DBID = GE

# ＧＵＩリソース
# RESOURCE_GUI = resource.GUIResource
#XX RESOURCE_GUI_DB = SELECT
GUIKEY,ADDRESS,SEQUENCE,CLASSIFY,NAME,LONGNAME,ROLE,USERGROUP,PROJECT,RWMODE,TA
RGET FROM GE11 WHERE SYSTEM_ID = ? AND LANG = ?
RESOURCE_GUI_DB = SELECT
GUIKEY,ADDRESS,SEQNO,CLASSIFY,SNAME,LNAME,ROLES,'','',RWMODE,TARGET,KBLINK FROM
GE11 WHERE SYSTEM_ID in ( ?,'GE' ) AND LANG = ? AND FGJ = '1' ORDER BY GUIKEY,KBSAKU
# RESOURCE_GUI_DBID = GE

# ユーザーリソース
# RESOURCE_USER = resource.UserResource
#XX RESOURCE_USER_DB = SELECT
USERID,PASSWD,LANG,NAME_JA,NAME_EN,MAILTO,MAILUSERID,MAILPASSWD,ROLE,USERGROUP,
PROJECT FROM GE10 WHERE SYSTEM_ID = ?
RESOURCE_USER_DB = SELECT USERID,PASSWD,LANG,NAME,'','','','',ROLE,'','' FROM GE10
WHERE SYSTEM_ID in ( ?,'GE' )
# RESOURCE_USER_DBID = GE
```

```
# カラムリソース
# RESOURCE_COLUMN = resource.DBColumnResource
RESOURCE_COLUMN_DB = SELECT
COLUMN_NAME,TYPE,USE_LENGTH,'TRUE',RENDERER,EDITOR,COLUMN_TYPE,DATA_DEFAULT,LAB
EL_CLM,CODE_CLM,RENDERER_PARAM,EDITOR_PARAM,TYPE_PARAM FROM GE03 WHERE SYSTEM_ID
in ( ?,'GE' ) ORDER BY COLUMN_NAME,KBSAKU
# RESOURCE_COLUMN_DBID = GE
```

4. 各種リソース定義

各種リソースファイルと、その概要を以下に説明します。

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 カラムリソース

カラムリソースは、データベース等のカラムIDをキーに、ラベルID、レンデラー、エディター、DBタイプ等を指定して、カラムオブジェクトを構築します。
カラムオブジェクトは、データベース検索結果の文字列(VARCHAR2)や数字(NUMBER)などのプリミティブ型情報を、オブジェクト的な振る舞い（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）を持たせることができます。カラムリソースは、国別にリソースを持っていません。
詳細は、『第9章 カラムオブジェクト』をご参照ください。

③ 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。

④ コードリソース

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 ・・・

この、コードリソースと、レンデラー、エディターが組み合わさることで、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ことができます。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、なお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。

⑤ 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。

⑥ 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画面のフォルダ名と連動しています（コラム参照）。
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ことが可能です。
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、なお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。

コラム ≪GUI リソースの画面 ID とフォルダの関係≫

GUI リソースの ID は、基本的にはそのアプリケーションの格納フォルダと同一 ID です。
しかし、場合によっては、画面名称を変えたり、制御を変えたい場合があり、その場合でもアプリケーション自身は、共通に使用したい場合があります。
その場合には、画面 ID を、アクセスされたフォルダ名から取得する現時点のエンジンの構造には無理があります。つまり、メニューやリンクから飛ばされた場合に、自分がどの画面 ID かを常に認識できる機能が必要ということです。
これは、異なる画面 ID で、同一フォルダにアクセスするケースです。

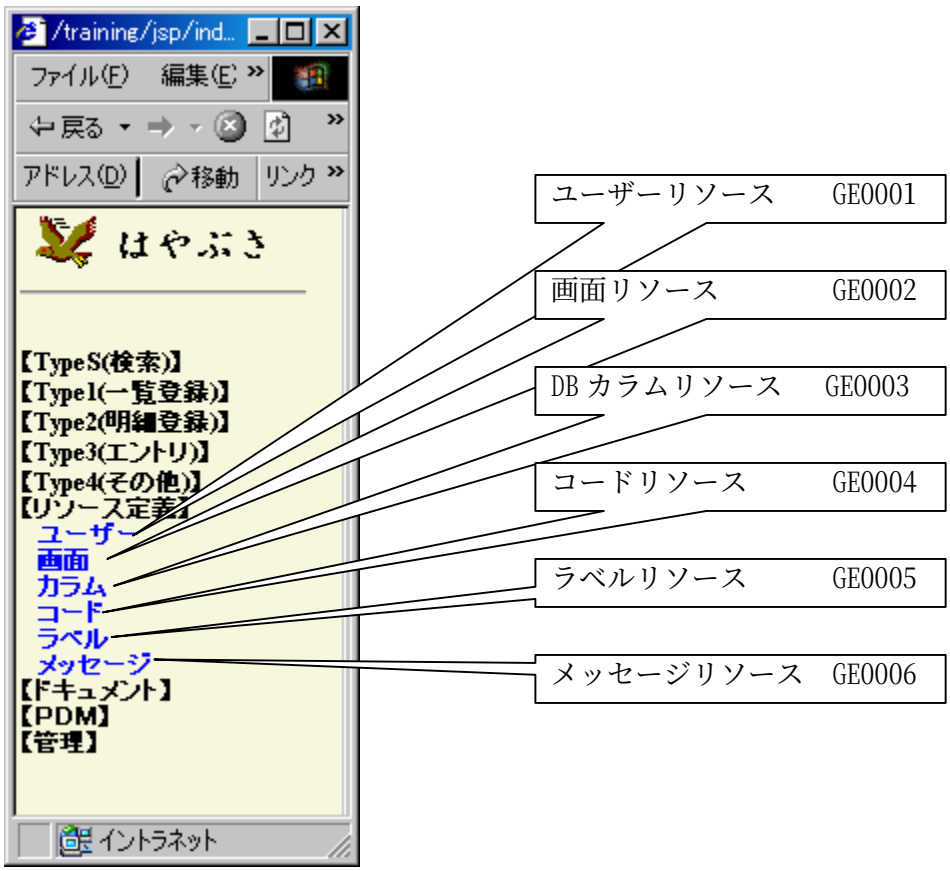
反対に、機能拡張や顧客対応の場合など、画面 ID は同一（つまり、メニューやリンクの値）でも、飛び先のフォルダを分けたい場合は、同一画面 ID から異なるフォルダへのアクセスというケースになります。

この両方の機能は、残念ながら現時点のエンジンでは未対応になっています。

第5章 リソース管理ツール

1. リソース DB 管理ツール

リソースファイルは Web アプリケーションで管理しています。ここでは、その詳細について説明します。



各リソースファイルを更新すると、キャッシュがクリアされ、リアルタイムで反映されます。

管理 ｜ リソース管理画面－プール削除との違いは、プール削除では、データベースのコネクションプーリングも、削除することです。

2. Tomcat Manager アプリケーション

Tomcat 4.1 より、従来のマネージャ機能(コマンドのみ)から、Web 上で操作できるようになりました。ここでは、この Manager アプリケーションの設定と、使い方を説明します。

① tomcat¥conf/tomcat-users.xml ファイルの編集、認証データベースの登録

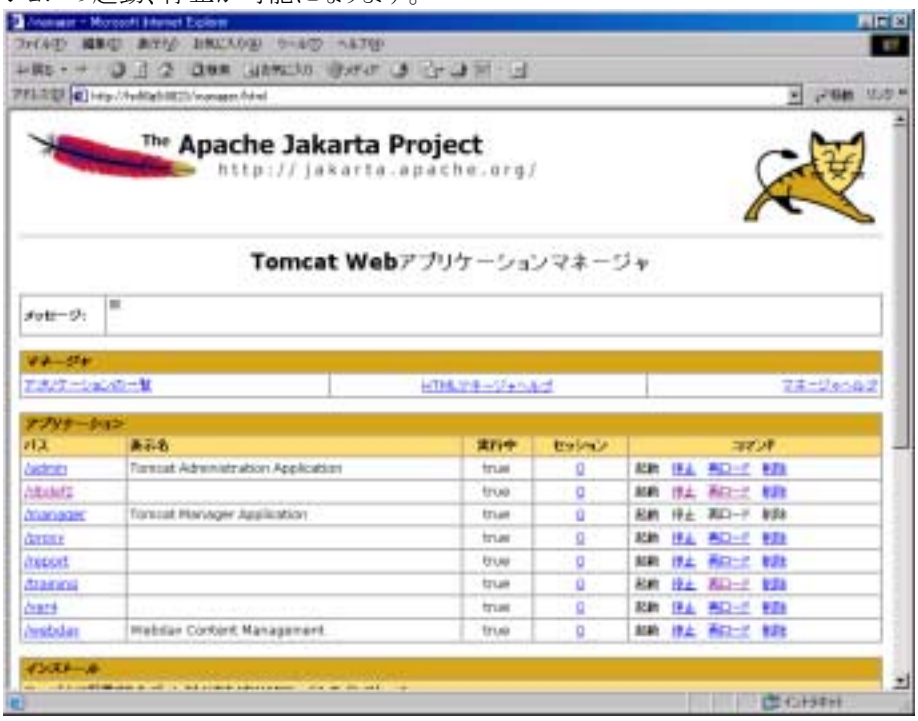
Manager アプリケーションを使用するには、manager ロールをもつユーザーが必要です。メモリレルムなら、tomcat-users.xml ファイルを、認証 DB なら、manager ロールのユーザーを先に登録しておく必要があります。

② tomcat¥conf/server.xml ファイルの編集

```
<Context path="/manager"
         docBase="H:\java\tomcat4.1.27\server\webapps\manager"
         debug="0" privileged="true">
    <Logger className="org.apache.catalina.logger.FileLogger"
        prefix="localhost_manager_log." suffix=".txt"
        timestamp="true"/>
</Context>
```

③ http://サーバー名:8823/manager/html でアクセス

このマネージャ画面を利用して、Tomcat の再起動なしで、個々の Web アプリケーションの起動、停止が可能になります。



3. Tomcat Admin アプリケーション

Tomcat 4.1 より、server.xml や tomcat-users.xml ファイルの設定を、Web アプリケーションから行うことができるようになりました。
ただし、tomcat-users.xml ファイルでのユーザー管理は、通常のエンジンを用いるアプリケーションでは行いません。(独自のデータベースで認証するため)
また、server.xml は、コメントなど、削除されてしまい、あまり使い勝手が良くありません。
ここでは、あくまで設定方法を述べるにとどめます。設定方法は、Manager アプリケーションの設定と、ほとんど同じです。

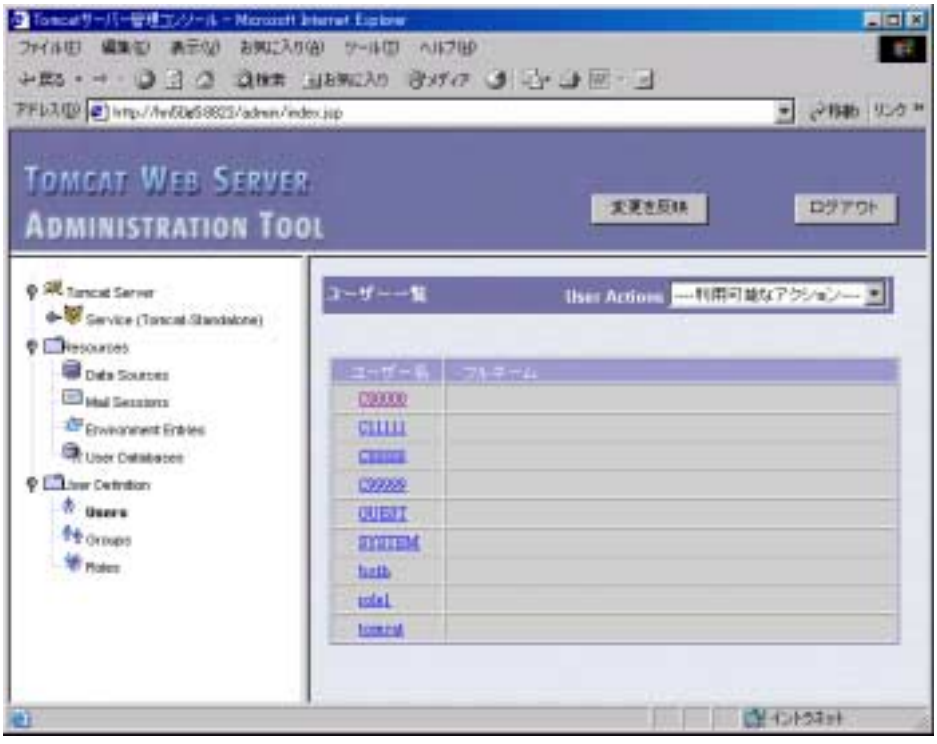
① tomcat¥conf/tomcat-users.xml ファイルの編集、認証データベースの登録

admin ロールを持ったユーザーを作成する必要があります。

② tomcat¥conf/server.xml ファイルの編集

```
<Context path="/admin"
         docBase="H:\java\tomcat4.1.27\server\webapps\admin"
         debug="0" privileged="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
          prefix="localhost_admin_log." suffix=".txt"
          timestamp="true"/>
</Context>
```

③ http://サーバー名:8823/admin/index.jsp　でアクセス



第 III 部 Webアプリケーション・セキュリティ

『年寄りは若い者に貯金をしろと言うが、それは間違っている。
最後の一銭まで貯めようなどと考えたらいけない。自分に投資しなさい。
私は４０才になるまで、１ドルたりとも貯金をしたことがない。』

ヘンリー・フォード（フォード創業者）

ここでは、Webアプリケーションのセキュリティーに関する説明をします。
構成は、次のとおりです。

第 6 章　コンテナによるユーザー認証の設定
ユーザー認証の実現方法とその設定と、アクセス制限について、説明します。

第 7 章　ユーザー権限とロールの管理
ユーザー権限とロールの管理に関するすべての情報が含まれます。権限とロールの付与および取消しの方法について説明します。

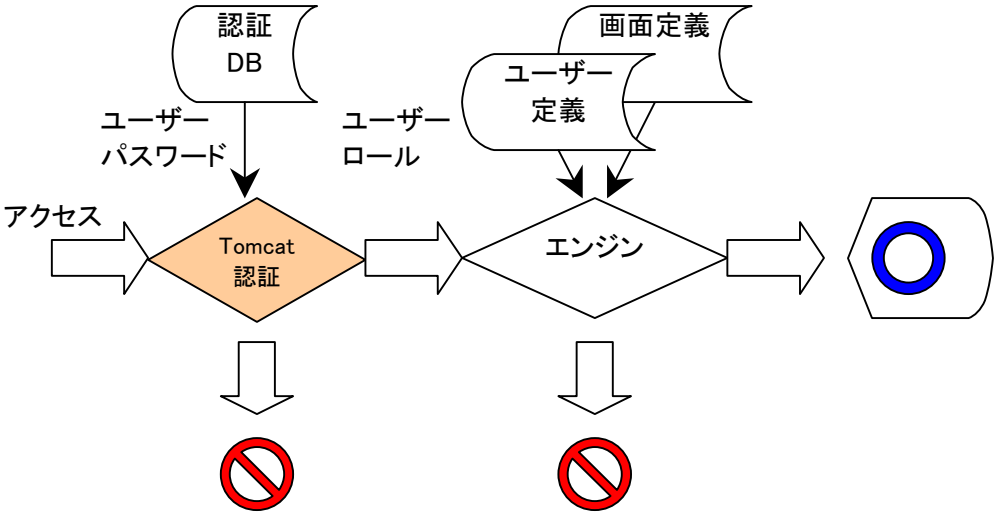
第 8 章　Webアプリケーションの稼動状況監視
稼動状況の監視について、ツールの使用方法を交えながら説明します。

第6章 コンテナによるユーザー認証の設定

この章では、ユーザー認証の実現方法とその設定と、アクセス制限について、説明します。

エンジンを用いた Web アプリケーションでは、2段階のセキュリティ機構を採用しています。まず、Tomcat によるユーザー認証により、そのシステムにログイン可能かどうかを判断し、次に、ユーザーロールと画面ロールによる各画面のアクセス許可を判定しています。

以下に、Tomcat（サーブレットコンテナ）による認証設定について、説明します。



1． tomcat-users.xml を用いた、メモリレルムの設定

Tomcat のデフォルトである、メモリレルムは、tomcat¥conf¥server.xml ファイルに記述されています。このファイルは、ユーザー認証に、tomcat-users.xml を使用します。
Tomcat 4.1 では、デフォルトは、UserDatabase になっていますが、同じ tomcat-users.xml ファイルを利用して、認証します。

$APPS¥java¥tomcat4.1.27¥conf¥server.xml ファイル

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
           debug="0" resourceName="UserDatabase" />

<!--
<Realm className="org.apache.catalina.realm.MemoryRealm" />
-->
```

$APPS¥java¥tomcat4.1.27¥conf¥tomcat-users.xml ファイル

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="tomcat" password="tomcat"  roles="tomcat"/>
  <user username="role1"  password="tomcat"  roles="role1"/>
  <user username="C00000" password="C00000"  roles="admin,manager"/>
  <user username="SYSTEM" password="MANAGER" roles="manager"/>
</tomcat-users>
```

2. データベースを用いた、JDBC レルムの設定

テスト以外では、tomcat-users.xml ファイルを用いた認証は使用しません。もっとも一般的なのは、データベースを用いた、JDBC レルムでの認証です。

Tomcat では、users テーブル(user_name , user_pass)と、user_roles(user_name , role_name)という、2つのテーブルを用いて認証を行います。これらのテーブル名、カラム名は、自由に設定できますが、キー(user_name)は、同一の名称にしておく必要があります。

以下に、デフォルトの設定を示します。

$APPS¥java¥tomcat4.1.27¥conf¥server.xml ファイル

```
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
       driverName="oracle.jdbc.driver.OracleDriver"
       connectionURL="jdbc:oracle:thin:@ntserver:1521:ORCL"
       connectionName="scott" connectionPassword="tiger"
       userTable="users" userNameCol="user_name" userCredCol="user_pass"
       userRoleTable="user_roles" roleNameCol="role_name" />
```

これを、実際のアプリケーションの設定ファイルを示します。

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

このように、users テーブルと、user_roles を同じテーブルで兼用しています。Web アプリケーションでは、ユーザーがそのシステムにログインできるかどうかをこの認証で判断しているだけなので、ロールは、システム ID になります。

3. 各アプリケーション側の設定

先の設定は、Tomcat コンテナ側の設定です。個々のアプリケーションは、認証をする/しないを、個別に設定する必要があります。設定は、配備記述子(web.xml)で、設定します。

$UAP¥webapps¥システム ID¥WEB-INF¥web.xml ファイル

```
<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>GE</role-name>
  </auth-constraint>
</security-constraint>

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

url-pattern：認証対象となる URL を記述します。
http-method：認証対象となるメソッドを記述します。
role-name　：認証対象となるロールを記述します。先のシステム ID に合わせます。
　　* を設定すると、すべてのロールを許可します。
auth-method：認証方式を設定します。BASIC,DIGEST,FORM など

コラム ≪ダイジェスト認証とダイジェストパスワード≫

各アプリケーションの web.xml で認証方式を設定しますが、そこに DIGEST 認証方式を指定するとどうなるでしょうか？

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

この場合、サーバーよりワンタイムキーを受け取り、クライアントのブラウザが、パスワードをダイジェスト（例えば、MD5 ハッシュ）してサーバーに送ります。これを、サーバー側が同様の手順でダイジェストし比較するのですが、Tomcat にはこの方式が使えません。

Tomcat でレルムに、digest="MD5" と設定すると、どういう動きをするのでしょうか？

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
       debug="0" resourceName="UserDatabase" digest="MD5" />
```

これは、Tomcat の tomcat-users.xml ファイルのパスワードを、**あらかじめ** MD5 でダイジェストしておき、クライアントからのパスワードを Tomcat 側でダイジェストし、このパスワードと比較します。 たしかに、パスワードファイルを直接見られた場合でも、問題なくなりますが、ネットワーク上には、**ほとんど生のパスワード情報**（BASE64 でエンコードされているが）が流れています。

```
<tomcat-users>
    <user username="tomcat"
          password="1b359d8753858b55befa0441067aaed3"
          roles="tomcat"/>
</tomcat-users>
```

どうしても、ネットワーク上のデータまで見られたくない場合は、SSL 等と組み合わせる必要があります。ちなみに、Tomcat には指定の文字列をダイジェストするコマンドファイル（バッチファイル）が用意されています。

```
%CATALINA_HOME%¥bin¥digest.bat -a md5 tomcat

tomcat:1b359d8753858b55befa0441067aaed3
```

と、なります。

コラム　≪Web エンジンでのダイジェストパスワード≫

パスワードファイルの場合は、先のコマンドで一人ずつ設定すれば良いのですが、Web アプリケーションでJDBCRealmを利用する場合、データベースにダイジェストパスワードを登録する必要があります。カラムリソースで、ダイジェストにしたいカラム（例えば、PASSWD）の DBType属性にMD5 を指定することで、データベース登録時にダイジェスト化して登録することが可能になります。この場合、再度読み出して登録すると、ダイジェスト化されたパスワードが再びダイジェスト化されますので、2度とログインできなくなる恐れがあるため、ご注意ください。

4. 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>
```

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.coyote.tomcat4.CoyoteConnector"
               port="8443" minProcessors="5" maxProcessors="75"
               enableLookups="false"
               acceptCount="100" debug="0" scheme="https" secure="true"
               useURIValidationHack="false" disableUploadTimeout="true">
     <Factory className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
              clientAuth="false" protocol="TLS"
              keystoreFile="C:\Documents and Settings\C87019.DOM_MIS1/.keystore"
              keystorePass="tomcat" />
    </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>GE</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

```
D:¥>%JAVA_HOME%/bin/keytool -genkey -alias tomcat -keyalg RSA
キーストアのパスワードを入力してください:  tomcat
姓名を入力してください。
  [Unknown]:  hn50g5.mis.muratec.co.jp
組織単位名を入力してください。
  [Unknown]:  hayabusa
組織名を入力してください。
  [Unknown]:  mis
都市名または地域名を入力してください。
  [Unknown]:  Kyoto
州名または地方名を入力してください。
  [Unknown]:  Kyoto
この単位に該当する 2 文字の国番号を入力してください。
  [Unknown]:  ja
CN=hn50g5.mis.muratec.co.jp, OU=hayabusa, O=mis, L=Kyoto, ST=Kyoto, C=ja
でよろしいですか?
  [no]:  yes

<tomcat> の鍵パスワードを入力してください。
        (キーストアのパスワードと同じ場合は RETURN を押してください):

D:¥>
```

6. シングル・サインオン

Tomcat の SingleSignOn バルブを利用可能にします。これは、同一ホスト上の異なるアプリケーション間で、1回のログインで認証情報を再利用できる(何度もログインしなくてすむ)機能です。通常、BASIC 認証であれば、ブラウザがキャッシュするため、シングルサインオン状態になりますが、WEB-INF/web.xml の login-config の realm-name に、それぞれ、アプリケーションごとの名称を設定している場合は、 異なる領域であると判断されるため、ブラウザでも別々にキャッシュされるため、アプリケーションごとに、認証ダイアログが立ち上がります。
SingleSignOn バルブを有効にしておけば、(処理時間は増えますが)そのような設定でも、シングルサインオンになります。

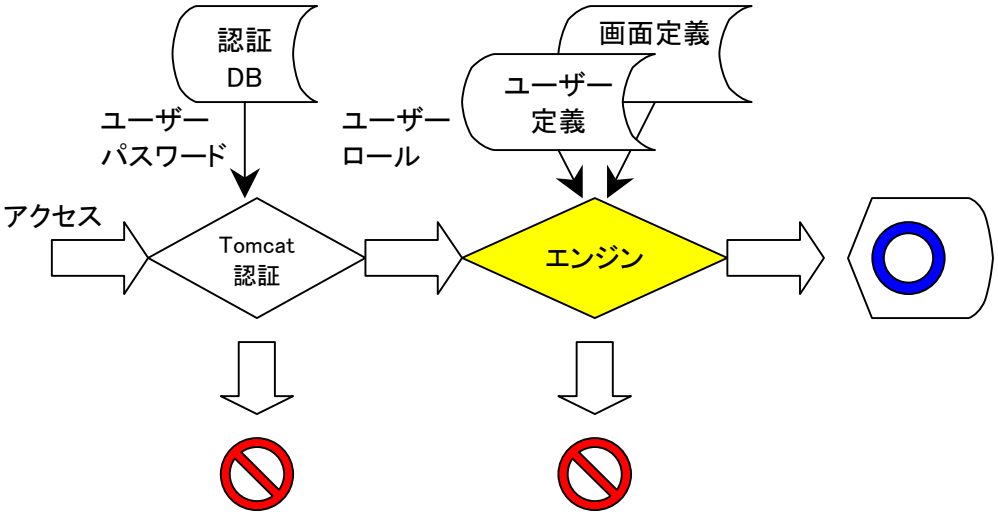
$APPS¥java¥tomcat4.1.27¥conf¥server.xml ファイル

```
<Valve className="org.apache.catalina.authenticator.SingleSignOn"
       debug="0"/>
```

第7章　ユーザー権限とロールの管理

エンジンを用いた Web アプリケーションでは、2段階のセキュリティ機構を採用しています。まず、Tomcat によるユーザー認証により、そのシステムにログイン可能かどうかを判断し、次に、ユーザーロールと画面ロールによる各画面のアクセス許可を判定しています。

この章では、ユーザー権限とロールの管理に関するすべての情報が含まれます。権限とロールの付与および取消しの方法について説明します。



1. ユーザーリソースと、ユーザー権限

ユーザーリソースは、ユーザー定義データベースより作成されます。これは、システムIDごとにユーザーを設定することができます。ログイン認証時には、ロールテーブル（ロールとは、SYSTEM_ID）として、ログイン後には、システムごとのユーザー権限を与えるテーブルとして使用されます。

GE10

	ユーザー定義		
○	システム ID	SYSTEM_ID	X(10)
○	ユーザーID	USERID	X(10)
	パスワード	PASSWD	X(32)
	ロケール(言語)	LANG	X(2)
	名称	NAME	X(40)
	ロール	ROLE	X(20)
	共通項目		

ユーザー定義のロールは、唯一の権限としてシステムごとに設定します。
つまり、ユーザーAさんが、課長で技術管理部で、なおかつ出図承認者であれば、それぞれ、3つのロールを与えるというのではなく、『課長＋技術管理部＋出図承認者』というロールを与えます。

これは、ユーザー権限の確認ロジックの簡素化のための処置です。

ユーザーのロールには、3つの特別なロールがあらかじめ設定されています。

- root
- manager
- admin

これらのロールは、システム的に予約されており、すべての画面に対してのアクセス権を持っています。先に述べました、manager アプリケーション、admin アプリケーションは、それぞれ、manager と、admin という特別なロールが必要です。 manager と、admin 両方のアプリケーションを使用したい場合は、各アプリケーション側の web.xml ファイルの設定で、対応してください。

2. GUIリソースと、画面アクセス制御

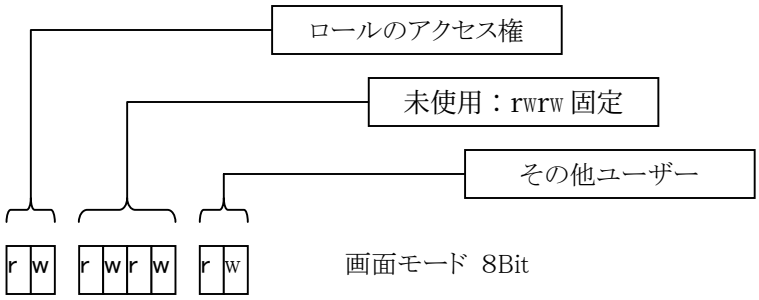
GUIリソースは、画面定義データベースより作成されます。これは、システムIDごとに画面IDを設定することができます。画面に対するアクセス権は、ユーザーロールと、画面ロールズ、そして、モードで制御します。

GE11

	画面定義		
○	システム ID	SYSTEM_ID	X(10)
○	ロケール(言語)	LANG	X(2)
○	画面 ID	GUIKEY	X(30)
	アドレス	ADDRESS	X(128)
	表示順	SEQNO	S9(5)
	分類	CLASSIFY	X(30)
	名前(短)	SNAME	X(60)
	名前(長)	LNAME	X(120)
	ロールズ	ROLES	X(200)
	モード	RWMODE	X(8)
	ターゲット	TARGET	X(10)
	リンク区分	KBLINK	X(10)
	作成区分	KBSAKU	X(1)
	共通項目		

画面ロールズは、ユーザーロールを、"|" で区切った文字列で、表されます。これは、ユーザーロールが、画面ロールズに含まれているかどうかで、判断しています。

画面モードは、それぞれの画面に対するアクセス権を、8Bit の rw 属性を用いて設定します。
各ビット部分は、リード属性には、"r" , "-" が指定でき、ライト属性には、"w" , "-" が指定できます。"-" は、アクセス不許可を表しています。
※ モードの各項目はビットです。"r" = 1 , "-"=0 に対応させれば、判りやすいでしょう。



- リード属性
  リード属性は、読み取り許可を与えます。読み取り許可とは、メニューに表示される画面のことです。ここに、"-" を設定すると、メニューに表示されません。
  ただし、画面のアクセスを禁止するのではなく、直接 URL を叩く(つまり、他の画面から呼び出される)ことは、あります。

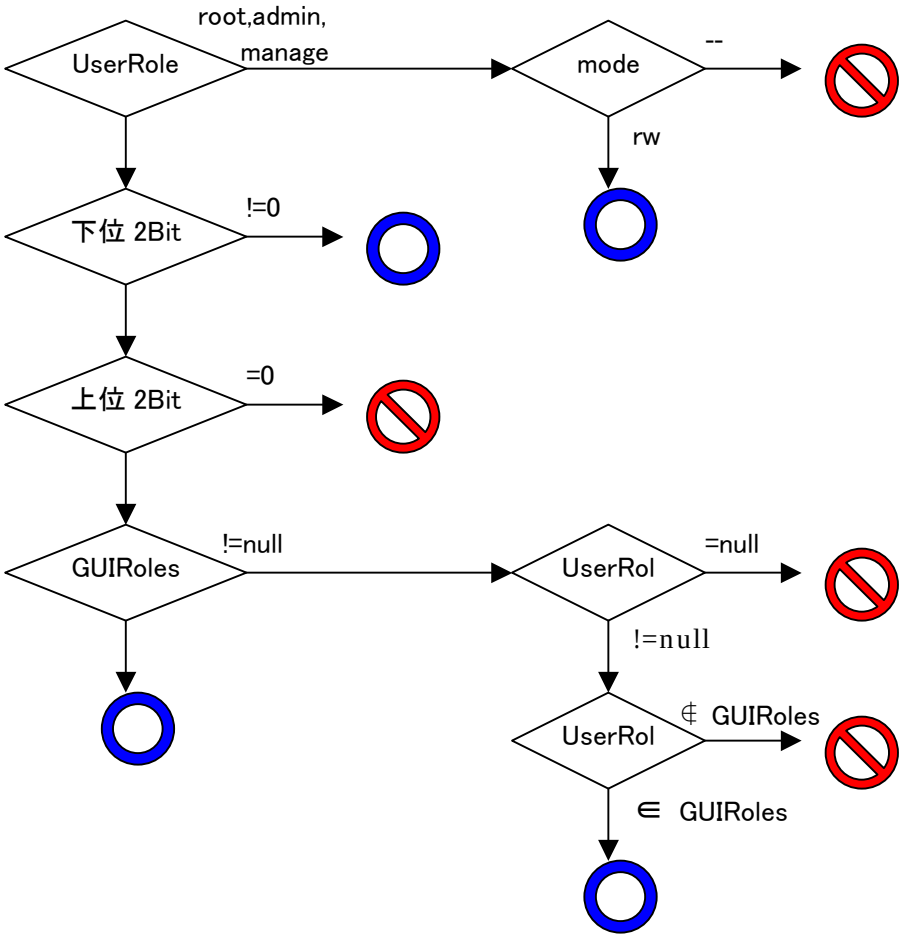
- ライト属性
  ライト属性は、書き込み許可を与えます。書き込み許可とは、登録系の SUBMIT ボタンを表示するということです。これは、writeCheck タグで囲まれたボタンは、このライト属性のあるユーザーにしか、見えないということです。書き込みを不許可にしたい場合は、ここに、"-" を設定してください。

警告 ≪未使用部分の画面モードについて≫

画面モードの未使用部分は、リソースファイルでは、グループ、プロジェクトの属性として現在でも使用されています。これは、ロール、グループ、プロジェクト の各許可が、AND で連結されるため、これらをフルに使用すると非常に複雑な制御になってしまいます。そこで、リソース DB 化にともない、ロールのみで判断するように変更しました。

3. 画面アクセス許可のフロー

画面アクセス許可には、ユーザーロール、画面ロールズ、画面モードが関係しています。これらのチェック基準のフローを示します。



非常に複雑なロジックになっていますが、

・UserRole が root,admin,manager の場合は、画面モードに、一箇所でも rw があれば、許可
・下位2Bit（その他のユーザー）が、 rw ならば、許可
　※ つまり、無許可ユーザー（例えば GUEST 等）に画面を見せたくない場合は、
　　下位2Bit を -- にします。逆に、検索のみ許可する場合は、r-, すべて許可する
　　場合は、 rw を設定します。
・上位2Bit が、-- ならば、不許可
・画面ロールズになにかが記載されており、ユーザーロールにも何かが記載されている場合、
ユーザーロールが画面ロールズに含まれていれば、許可。含まれていなければ不許可。
　※ つまり、画面ロールズか、ユーザーロールになにも設定されていない場合は、
　　全て許可したと判断します。

第8章 Webアプリケーションの稼動状況監視

この章では、稼動状況の監視について、ツールの使用方法を交えながら説明します。

1. 監視項目

Web アプリケーションを運用するにあたり、そのシステム情報と稼動状況を監視することは、非常に重要です。監視項目は、次のとおりです。

システム情報
- ① JDK の種類、バージョン
- ② Tomcat の種類、バージョン
- ③ メモリ使用量

稼動状況
- ④ ログインユーザー数
- ⑤ JDBC プール数

2. システム情報の設定

システムをインストール又は、立ち上げた場合に、初期設定されます。

① JDK の種類、バージョン
② Tomcat の種類、バージョン

上記2つは、インストールしたアプリケーションに依存します。しかし、たまにバージョン違いの JDK や Tomcat を同じOS上にインストールした場合は、実際に稼動しているバージョンを確認するのは、難しくなります。
これらの情報を初期設定しているファイルは、UAP¥bin¥init.bat ファイルで、指定したアプリケーションが使用されます。場合によっては、ひとつのサーバーに、異なる Tomcat が、立ち上がっているかもしれません。そのような場合でも、Web アプリケーションのツールにより確認できれば便利です。

UAP¥bin¥init.bat ファイル

```
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
```

③ メモリ使用量

メモリ使用量は、システム起動時に確保されます。初期値と最大値をオプションで指定できます。稼動後の実使用メモリは、動的に変化します。

```
set CATALINA_OPTS=-server -Xms128m -Xmx128m -Xss256k
```

3. 稼動状況

④ ログインユーザー数

ログインユーザー数に、制限は設けていません。もちろん、物理的に、一人ログインするたび、リソースが消費されます。無制限にログインできるはずはありません。
ログインユーザー数に影響するのは、メモリリソースです。これは、1ユーザー1セッションを使用し、セッション情報に各種検索データやユーザー情報などをキャッシュするためです。
セッションは、1ユーザー単位ではなく、1回のブラウザでのアクセスごとに作成されます。つまり、同一人物がブラウザを閉じて、再びログインすると、先のセッションが残ったまま、新たなセッションが作成されます。
この、セッションの有効期間は、src¥resource¥SystemResource.properties ファイルで、設定しています。

```
# セッションタイムアウト(秒)
MAX_INACTIVE_INTERVAL = 900
```

初期値(推奨値)として、15分です。これは、クライアントからのアクセスがなくなってからの時間です。(連続使用している間は、クリアされていきます。)
エンジンでは、セッションに検索結果を蓄えている関係で、操作途中でセッションが無効になった場合は、次回アクセス時にエラーを返しています。
それ以外(メニューからアクセス、再度検索するなど)では、セッションを新たに作り直して、ユーザーからは連続していたかのように動作することができます。
(もちろん、それで影響がないように、アプリケーションを組む必要があります。)

⑤JDBC プール数

JDBC プールは、Web アプリケーションごとに作成される、コネクションプールです。ログインユーザー数には影響ありませんが、もちろん、同時アクセスが増えれば、それだけプールに多くのコネクションを用意しておく必要があります。
設定は、src¥resource¥SystemResource.properties ファイルで行います。

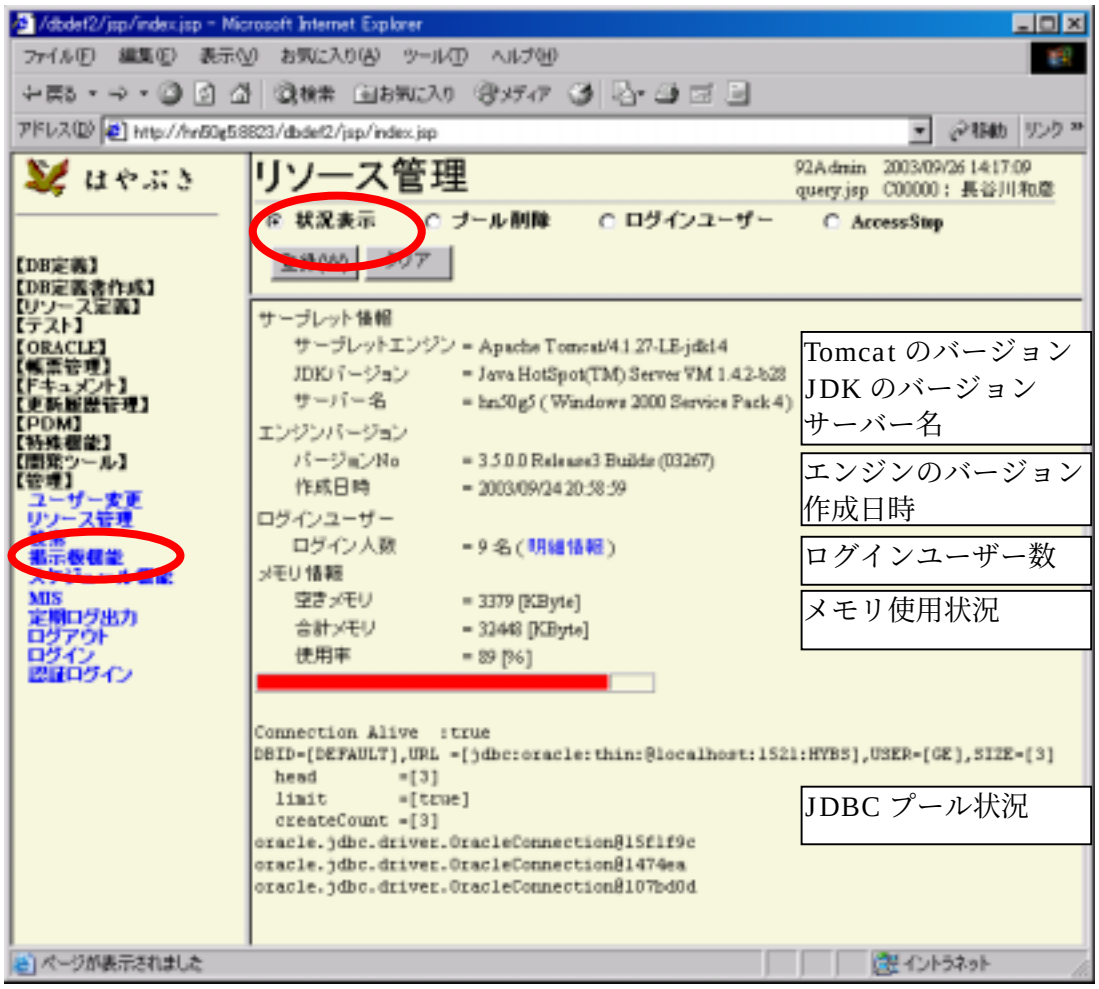
```
# ConnectionFactory.java
#
# DB_MAXCOUNT 最大プール数 は、コネクションプールの最大数です。
# この最大数が、同時接続がこの数を超えると、MissingResourceException が、
# 投げられますので、後程再接続する必要があります。
#
# DB_MINCOUNT コネクションプールの最小数 は、予めコネクションを準備しておきます。

DEFAULT_DB_MINCOUNT   = 3
DEFAULT_DB_MAXCOUNT   = 10
```

この例では、初期値に 3 コネクションを作成します。そして、最大10コネクションまで順次増やしていきます。

4. 監視ツール

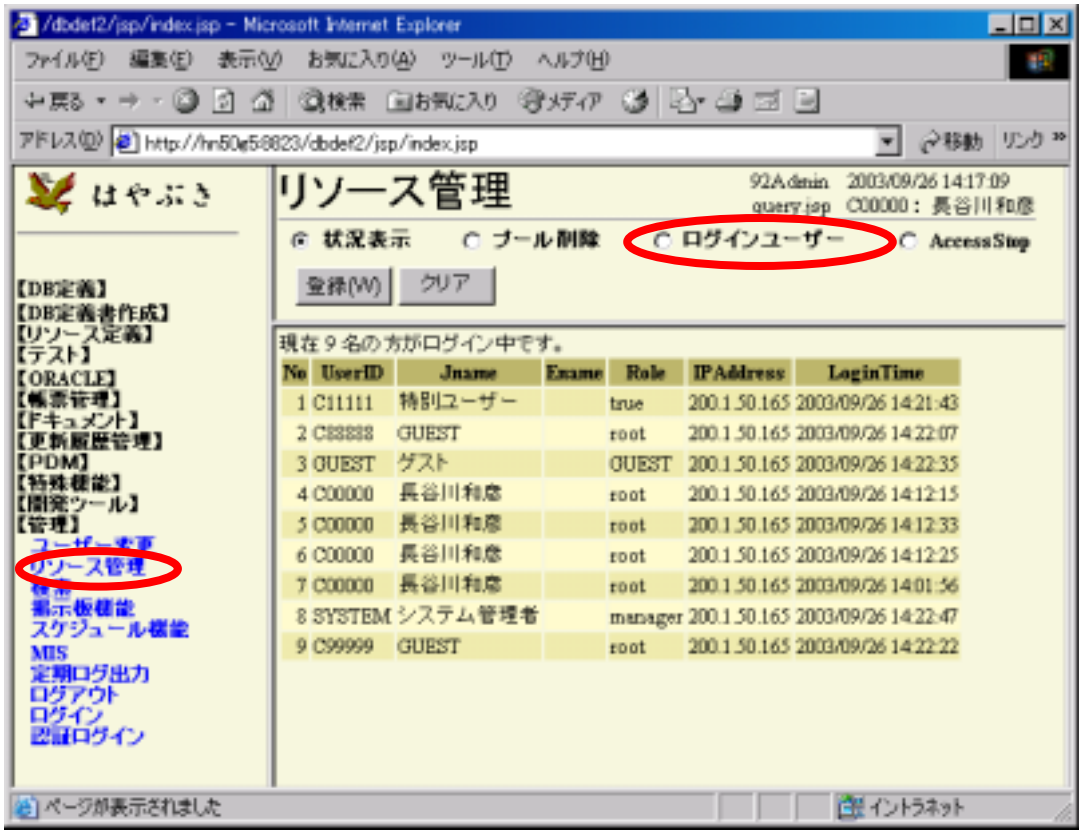
稼動状況の監視ツールとして、『92Admin』という画面を用意しています。
先にあげました監視項目を確認することができます。



ログインユーザーの明細情報から、現在ログインしているユーザーの詳細情報が確認できます。
この詳細情報は、リソース管理の『ログインユーザー』と、同じ画面です。

5. ログインユーザー

ログインユーザーは、現在生きているセッション情報と紐ついているユーザー情報を表示します。これは、同一ユーザーID であっても、ブラウザのセッション情報が異なる場合は、別ユーザーとして認識されます。
ブラウザを閉じずに、右ボタンメニューで開いたり、ポップアップダイアログなどを出した場合は、同一セッションになりますので、そのような運用方法で問題ないかは、アプリケーション側の作りに依存します。十分ご注意ください。

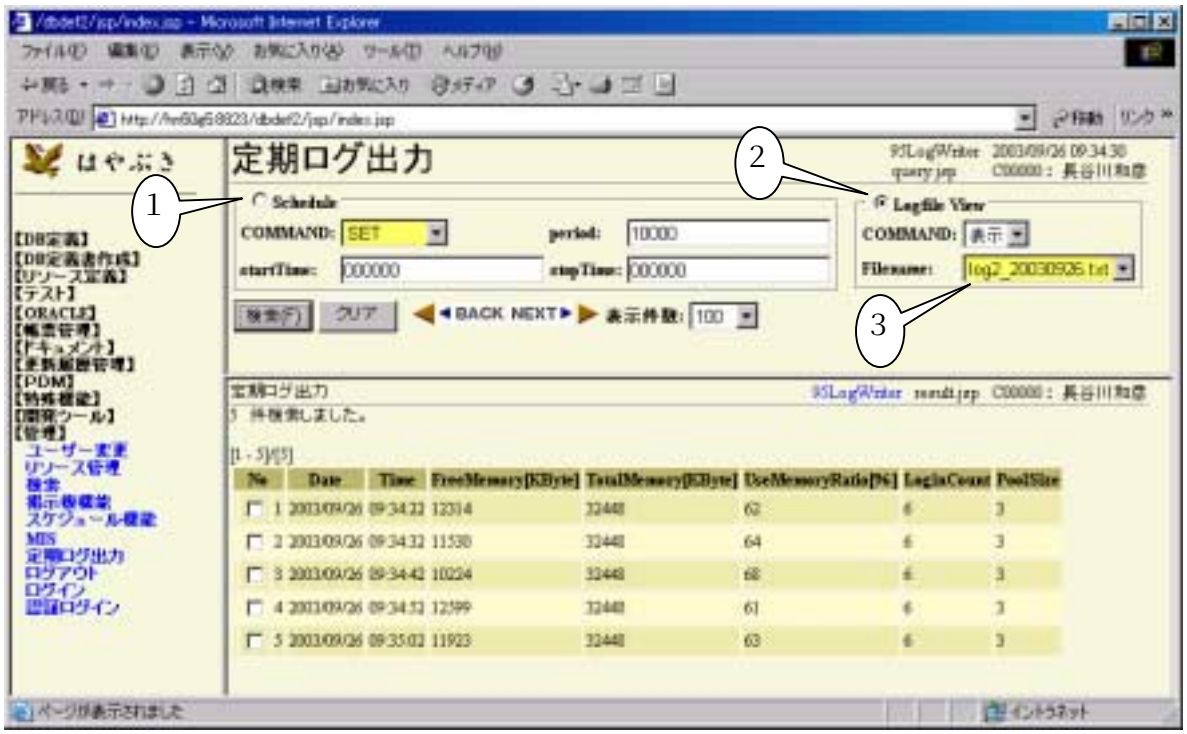


6. 定期ログ出力

先の管理画面は、ある一瞬の状況を把握しただけです。一日を通して、どのような状況になっているのか、ログインユーザー数やメモリ使用量、JDBCコネクタ数など、定期的にログファイルとしてサンプリングしたい場合があります。エンジンでは、『定期ログ出力』画面を用意しています。 これは、指定時間間隔で、定期的に JOB を実行し、LOG を採取します。LOG 内容は、空きメモリ、全メモリ、使用メモリ率(%)、ログインユーザー数、JDBC プール数です。

JDBC プール数は、最大数であり、その瞬間に使われている数ではありません。

※ 現 Version では、JDBC プールは、一度増えると2度と消えません。(最大数は超えません)
ただし、JDBC でエラーが発生した場合は、そのコネクションは、プールから削除します。
つまり、最大数(過去に作成した数)が減るということは、データベースエラーが発生したと判断することが可能です。
なお、JDBC プールを削除(初期化)するのは、リソース管理メニューの『プール削除』です。



使い方:

1. Schedule で、ログ出力デーモンを起動します。
2. ログを確認するには、LogfileView を選びます。
3. ログファイルは、日付けごとに作成されます。
4. EXCEL 出力等で、取り込み、グラフ化してください。

第 IV 部 パフォーマンスチューニング

『常に社員をほめるんだ。決して、けなしてはならない』

リチャード・ブランソン（ヴァージン創業者）

ここでは、パフォーマンスチューニングについて説明します。
構成は次のとおりです。

第 9 章　パフォーマンスチューニング
パフォーマンスチューニングについて説明します。

第 10 章　データベースアクセスレポート
ORACLE 監視ツールを用いて、データベースチューニングを行う方法を説明します。

第 11 章　ストレスツール（JMeter）
ストレスツールについて説明します。

第9章 パフォーマンスチューニング

この章では、パフォーマンスチューニングについて説明します。

1. チューニング対象

まず、パフォーマンスチューニングするにあたり、なにが影響しているのか、切り分ける必要があります。これは、チューニング対象により、チューニング方法が異なるためです。また、チューニング方法も、チューニング対象の状況（使われ方）により、変わってきます。つまり、チューニングを行うにあたり、チューニング対象を洗い出し、分類し、状況を把握した後、チューニング方法を決定するという手順を踏む必要があると言うことです。

チューニング対象を、洗い出してみましょう。

- OS（Windows のチューニング）
- JDK のチューニング
- ServletEngine(Tomcat)のチューニング
- Web エンジン（はやぶさ）のチューニング
- Web アプリケーション（JSP、PL/SQL）のチューニング
- データベース（ORACLE)のチューニング
- ネットワーク、ルータ等のチューニング
- クライアント（ブラウザ等）のチューニング

これらを、**要求パフォーマンスになるまで**、効果のあるところから、順につぶしていけば良いということになります。この、**要求パフォーマンスになるまで**というのが、実は非常に深い意味があり、逆に、要求パフォーマンスであれば、チューニングしなくて良いことになります。

もちろん、**『一般的な注意を守って構築したシステムであれば』**という前提条件がつきます。

2. チューニング一般

先に挙げましたチューニング対象について、一般的な注意点を列挙します。

- OS（Windows のチューニング）
  ここで、本格的な OS のチューニングを議論するつもりはありません。
  通常は、サーバー購入時に一般的なチューニングは行われているという前提で、
  考えればよいでしょう。
  つまり、サーバー購入時に、CPU クロック数、CPU 数、メモリ量を、いくらまで
  設定できるか、予算との兼ね合いになります。

- JDK のチューニング
  JDK は、各社色々な JVM を発表していますが、基本は、SUN の JDK1．4で構わないと思います。
  特に、JDK1.4.1、1.4.2 に対して行われたチューニングにより、相当パフォーマンスが向上しているため、問題がない限り、最新の JDK を利用すればよいでしょう。
  起動時オプションとして、$UAP¥bin¥init.bat で、各種パラメータを指定します。

  set CATALINA_OPTS=-server -Xms128m -Xmx128m -Xss256k
  ↑ Java 起動時使用メモリ
  Xmsは、初期ヒープサイズ
  Xmxは、最大ヒープサイズ

  初期ヒープサイズと最大ヒープサイズを同じにしておくと、ヒープ拡張時のメモリ割り当てに時間を取られる時間がなくなります。

  サーバー環境が、マルチ CPU でメモリも豊富であれば、下記パラメータも試してみる価値があります。（実際には、1行で記述してください。）

  set CATALINA_OPTS=-server -Xms512m -Xmx512m -Xss256k
  -XX:NewSize=256m -XX:MaxNewSize=256m -XX:+AggresiveHeap

- ServletEngine(Tomcat)のチューニング
  Tomcat のチューニングは、APPS¥java¥tomcat¥conf¥server.xml ファイルで、指定します。

  <Connector className="org.apache.catalina.connector.http.HttpConnector"
  port="8823" **minProcessors="5" maxProcessors="75"**
  enableLookups="false" redirectPort="8443"
  acceptCount="10" debug="0" connectionTimeout="60000"/>

  enableLookups ：IP アドレスからサーバー名への DNS 逆引きです。
  基本的には、クライアントの名称は、不要です。よって、false に設定すれば、パフォーマンスが向上します。

  minProcessors ：コネクタへのリクエストの処理スレッド数の初期値
  常に、一定以上のユーザーによりアクセスされる場合に、設定すると有効です。しかし、一般には、それほど大量のアクセスが一定してあるというケースは、まれなので、デフォルト(5)で、問題ないでしょう。

  maxProcessors ：コネクタへのリクエストの処理スレッド数の最大値
  処理スレッドの上限を規定します。これ以上の要求があれば、サーバーは処理をブロックします。この値が大きすぎると、大量のスレッドが、多くのネットワーク接続を行うため、性能が劣化しますが、少ないと、処理を捌ききれなくなります。最大でも、200くらいに設定を抑えましょう。

acceptCount　：空きの接続プロセスを待っている間に受け入れられる接続数
maxProcessors で処理し切れなかった場合に、この設定数までのリクエストをサーバー側が受け入れます。この値を上回った場合は、着信拒否されますので、できるだけ大きいほうが良さそうですが、あまり大きいと、OS が不安定になります。1リクエストの処理時間も関係しますので、負荷テストや使用状況を良く見て、必要であれば変えればよいでしょう。

- Web エンジン（はやぶさ）のチューニング
エンジンそのもののチューニングは、随時対応しています。
ここでは、外部設定ファイルによる設定個所を説明します。

$UAP¥webapps¥システム ID¥src¥resource¥SystemResource.properties ファイル

```
# ConnectionFactory.java
#  設定キーワード  は、URL,USER,PASSWD,最大プール数、初期接続数  です。

#DEFAULT_DB_URL    = jdbc:oracle:oci:@HYBS
DEFAULT_DB_URL    = jdbc:oracle:thin:@localhost:1521:HYBS
DEFAULT_DB_USER    = GE
DEFAULT_DB_PASSWD    = GE
DEFAULT_DB_MINCOUNT    = 3
DEFAULT_DB_MAXCOUNT    = 10
```

dbid_DB_MINCOUNT： コネクションプールの初期接続数
起動時に、JDBC コネクションをこの数だけ作成し、プールします。一定以上のユーザーが常にコネクションを取り合う状況では、多いほうが良いでしょうが、基本的には、それほどこの値を変更する必要はありません。
逆に、ユーザーが少ない場合、初期値を 1 に設定しても良いでしょう。

dbid_DB_MAXCOUNT： コネクションプールの最大数です。
JDBC のコネクションの最大数です。この数の接続だけ、同時アクセスできます。1回当りの検索/登録時間と、同時ユーザー数との関係になりますので、データベースが早ければ、それほど多く作成させる必要はありません。

以下、上記コネクションが、いっぱいになった場合の処理を規定しています。
以下の設定値は、基本的に変更する必要はありません。

$UAP¥webapps¥システム ID¥src¥resource¥SystemResource.properties ファイル

```
# ConnectionFactory.java
#  コネクションを破棄するのに,何回リトライするか
DB_CLOSE_RETRY_COUNT = 10


# ConnectionFactory.java
```

# コネクションを破棄するリトライに，何 MS の間隔でリトライを行うか
DB_CLOSE_RETRY_TIME = 1000

# JDBCTableModel.java
# コネクションを取得するのに,何回リトライするか
DB_RETRY_COUNT = 10

# JDBCTableModel.java
# コネクションを取得するリトライに，何 MS の間隔でリトライを行うか
DB_RETRY_TIME = 1000

# ConnectionFactory.java
# Connection をプールしておく最大時間（秒）
# この時間（秒数）以上プールされている Connection は物理的に接続を
# 終了させて新しく Connection を作成します。
DB_MAX_CONNECTION_POOL_TIME = 900

※ 現在、DB_MAX_CONNECTION_POOL_TIME は、実装されていません。

- Web アプリケーション（JSP、PL/SQL）のチューニング
  JSP でのチューニングで大切なのは、ユーザーの体感速度です。JSP⇒HTML に変換されたデータに、大量に繰り返し項目の値が設定されていると、クライアント側のブラウザで、表示速度に影響が出てきます。テーブルの様な繰り返し属性に、font や style を個別に設定すると、このような状況が発生します。通常は、td より、tr に、直接個別のタグに指定するより、CSS ファイルを利用するなど、データ量を減らすことも重要です。
  あと、画面に表示する場合に、レンデラー、リンク、マーカーが適用されます。
  pageSize が少ないほど、表示処理にかかる時間が軽減されます。
  また、検索の最大件数も重要です。現エンジンでは、すべてのデータを最大件数まで一旦読み込んでから、表示に移ります。最大件数も、必要な画面のみ、個別に設定するほうがよいでしょう。

  $UAP¥webapps¥システム ID¥src¥resource¥SystemResource.properties ファイル

  # セッションタイムアウト（秒）
  MAX_INACTIVE_INTERVAL = 900

  # JDBCTableModel.java
  # データ検索時の最大件数
  # この件数以上のデータは、物理的に取得できなくなります。
  DB_MAX_ROW_COUNT = 1000

```
# 画面上に一度に表示されるデータ件数
# ただし, HTMLTextField.java のみ, 1件ずつ表示に固定されています。
HTML_PAGESIZE = 100
```

PL/SQL でのチューニングは、ほとんど ORACLE チューニングの世界なので、ここではあえて述べません。以下のパラメータは、PL/SQL を初めとする接続処理時間の最大値を指定しています。バッチ系では、デフォルト180秒では少ないかもしれません。

```
# AbstractQuery.java
# データ検索時の最大処理制限時間
# この時間(秒数)以上 SQL が終了しない場合は, MISServiceException を発行します。
DB_MAX_QUERY_TIMEOUT = 180
```

- データベース(ORACLE)のチューニング
  データベースのチューニングは、書籍、参考資料等山のようにありますので、ここでは述べません。エンジンとしては、チューニング結果を判定するアプリケーション(画面)を用意しています。
  このアプリケーションに関しては、次章で取り上げます。

  Web エンジンを利用する上で、考慮すべき設定項目として、
  - CURSOR_SHARING
  - OPTIMIZER_MODE

  が、挙げられます。

  CURSOR_SHARING=EXACT(デフォルト)に設定すると、SQL 文の共有プールのライブラリキャッシュに対して、全く同一の SQL のみ同じと認識します。エンジンでは、SQL 文字列を動的に作成(例えば、where 条件など)して実行するため、検索条件が異なるだけで、別の SQL として、ライブラリキャッシュに追加、解析されます。
  CURSOR_SHARING=SIMILAR または、FORCE を指定すると、リテラル値をバインド変数に変換し、同じ SQL 文として、共有することが可能になります。

  OPTIMIZER_MODE は、CHOOSE,ALL_ROWS,FIRST_ROWS_n,FIRST_ROWS, RULE があります。デフォルトの CHOOSE は、統計が使用可能かどうかによって、コストベースまたはルールベースのどちらかのアプローチを選択します。
  エンジンでは、基本的に画面系アプリケーションであり、かつ、検索件数も、SystemResource.properties ファイルの DB_MAX_ROW_COUNT = 1000 で、制限しているため、FIRST_ROWS_n または、FIRST_ROWS が有効になる可能性があります。
  ただし、ほとんどのケースで絞り込みの結果、DB_MAX_ROW_COUNT を超えないならば、ALL_ROWS の方が早いケースもあります。よって、SQL 個別にヒントとして、指示するのが、妥当なのかもしれません。

ヒント ≪SIMILAR とFORCE の違い≫

SIMILAR と FORCE の違いは、実行計画を機能低下させることなく SIMILAR が類似する文に SQL 領域を共有させるという点です。CURSOR_SHARING を FORCE に設定すると、類似文に実行可能な SQL 領域を共有するように強制します。この方法には、潜在的に実行計画の機能を低下させる可能性があります。したがって、計画が最適なものではなくなる可能性があってもカーソル共有率の向上を優先する場合に、FORCE を最後の手段として使用してください。

警告
複雑な問合せを使用している場合は、DSS 環境で CURSOR_SHARING を FORCE に設定することはお薦めしません。また、CURSOR_SHARING が SIMILAR または FORCE に設定されると、スター型変換はサポートされません。

ALTER SYSTEM SET CURSOR_SHARING = SIMILAR;
ALTER SESSION SET CURSOR_SHARING = SIMILAR;

ヒント ≪OPTIMIZER_MODE を使用する場合≫
OPTIMIZER_MODE 初期化パラメータで、インスタンスに最適化アプローチを選択するためのデフォルト動作を設定します。初期化パラメータには次の値があります。

CHOOSE
オプティマイザは、統計が使用可能かどうかによって、コストベースまたはルールベースのどちらかのアプローチを選択します。これはデフォルト値です。
ALL_ROWS
オプティマイザは、セッション内の SQL 文に対して、統計の存在の有無にかかわりなくコストベースのアプローチを使用し、最高のスループット(最小のリソースを使用して文全体を完成させること)を目標として最適化します。
FIRST_ROWS_n
オプティマイザは統計の有無とは関係なく、コストベースのアプローチを使用し、最短応答時間で最初の n 行を戻すように最適化します。n は 1 、10 、100 、1000 です。
FIRST_ROWS
オプティマイザは、コストと経験則を組み合せて、最初の数行の高速配信のための最適な計画を見つけます。注意: CBO は、経験則を使用すると、経験則を適用しない場合に比
べ、コストがはるかに大きい計画を生成する場合があります。FIRST_ROWS は、下位互換性とプラン・スタビリティのためのものです。
RULE
オプティマイザは、SQL 文に対して、統計の存在の有無にかかわりなくルールベースのアプローチを選択します。

ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS;

- ネットワーク、ルータ等のチューニング
  ネットワーク、ルータは、チューニングというより、設定ミスがないかどうかを確認することが先決です。DNS へのアクセスが遅い、名前解決ができない、10Base と100Base が混在し、不要なハンドシェークが入っている、TCP-IP 以外のパケットが、大量に飛び交っている・・・など。
  もちろん、パケットサイズを変えるなどのテクニックも存在しますが、それによる影響が明確に判断できない（テストできない）ならば、弊害があることも考え、慎重に対応してください。

- クライアント（ブラウザ等）のチューニング
  状況次第というところです。
  例えば、クライアントが比較的良質なマシンの場合、負荷分散効果として、クライアント側が多少重くなっても、サーバー処理を軽くした方が良い場合があります。
  逆に、クライアントにお金がかけられないと、サーバーで処理して、クライアントでの処理を極力減らすようにすべきです。
  『Web アプリケーション（JSP、PL/SQL）のチューニング』でも述べた、データ量を減らす方法は、クライアントの負荷軽減ですし、JavaScript による見栄えの向上は、クライアント側リソースに余裕がある場合には効果があります。

第10章 データベースアクセスレポート

ORACLE データベースをチューニングするにあたり、データベースの状況を確認することは、非常に重要です。ここでは、ORACLE 監視ツールを用いて、データベースチューニングを行う方法を説明します。

1. データベース監視項目

ORACLE 監視ツールには、以下の8つのメニューが用意されています。これらのうち、パフォーマンスに関連するのは、下記太字の項目です。

・DB 稼動状況
　　データベースの設定チェック
・DB 領域監視
・オブジェクト監視
　　オブジェクトの Analyze 状況一覧
・リソース使用状況
・セッション切断
　　ライブラリキャッシュのSQL文
・EXPLAIN PLAN
・TKPROF
・統計情報収集

以下、個々の画面の使い方と、初期設定方法について、述べていきます。

コラム ≪今後の予定≫

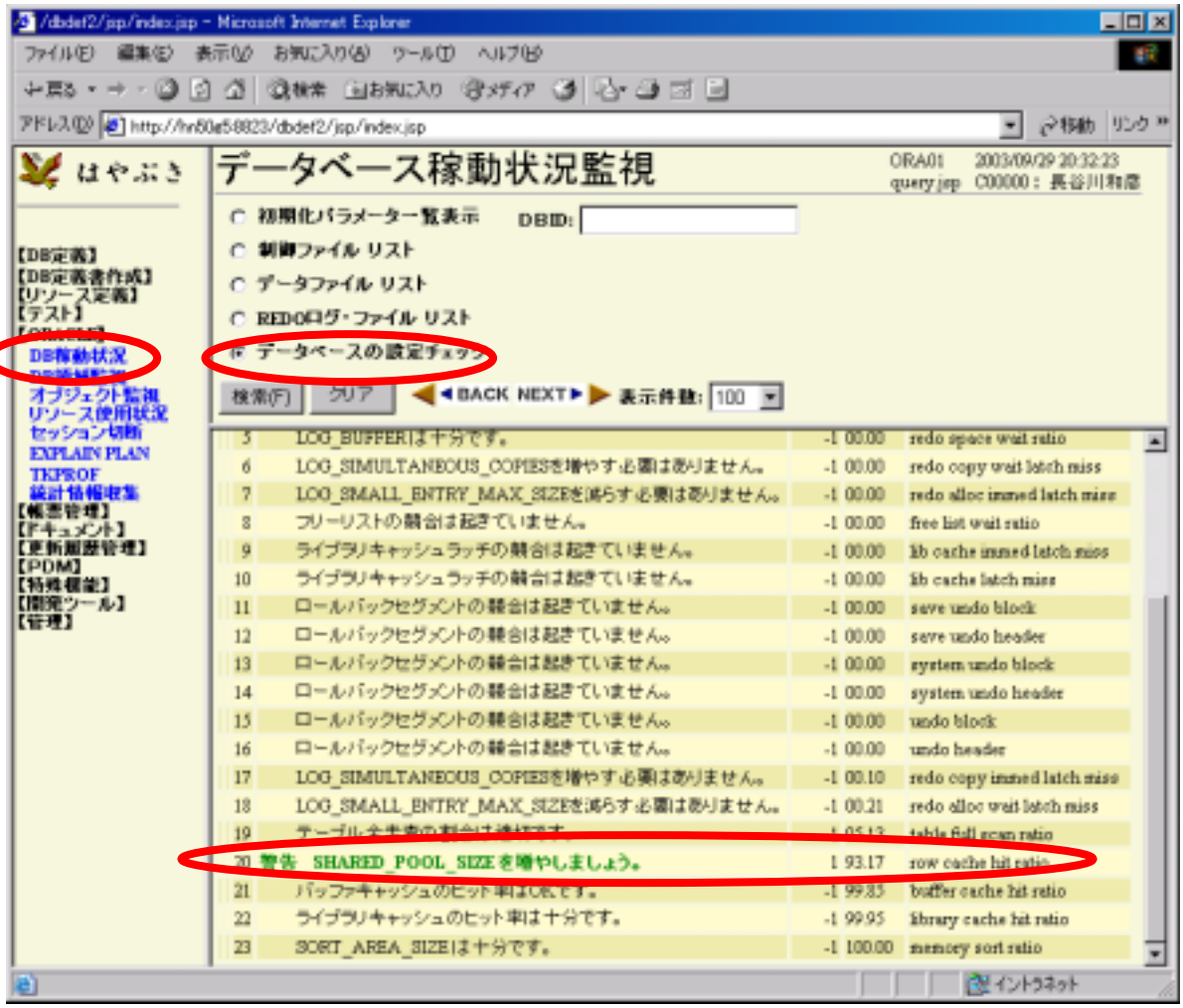
ORACLE には、STATSPACK 等のパフォーマンス診断ツールがあります。これらを簡易的に使用できるアプリケーションを順次作成していきたいと考えています。
さらに、自動診断や、不正な SQL を使用している画面のピックアップ、開発段階での SQL 文のアドバイス機能、運用開始後の調査、解析、レポートなど、色々考えられます。
ただし、あまり深入りすると、とんでもなく『あぶない世界 ※』であるため、費用対効果を常に意識しながら、進めていきたいと考えています。

※ あぶない世界 とは、我を忘れて没頭してしまい、本来の目的であるシステムを最適な状態で運用することを忘れ、ただただ、速さのみを追求するという人格破壊を招きかねない世界である・・・ということです。(^_^)"

2.　データベースの設定チェック

データベースの設定チェックは、各種データベース情報を検索し、一覧表示で設定状況の不具合を確認できます。これは、初期化ファイル（8i では、init.ora ファイル、9i では、バイナリ化されています）の設定に依存する項目のチェックに利用します。

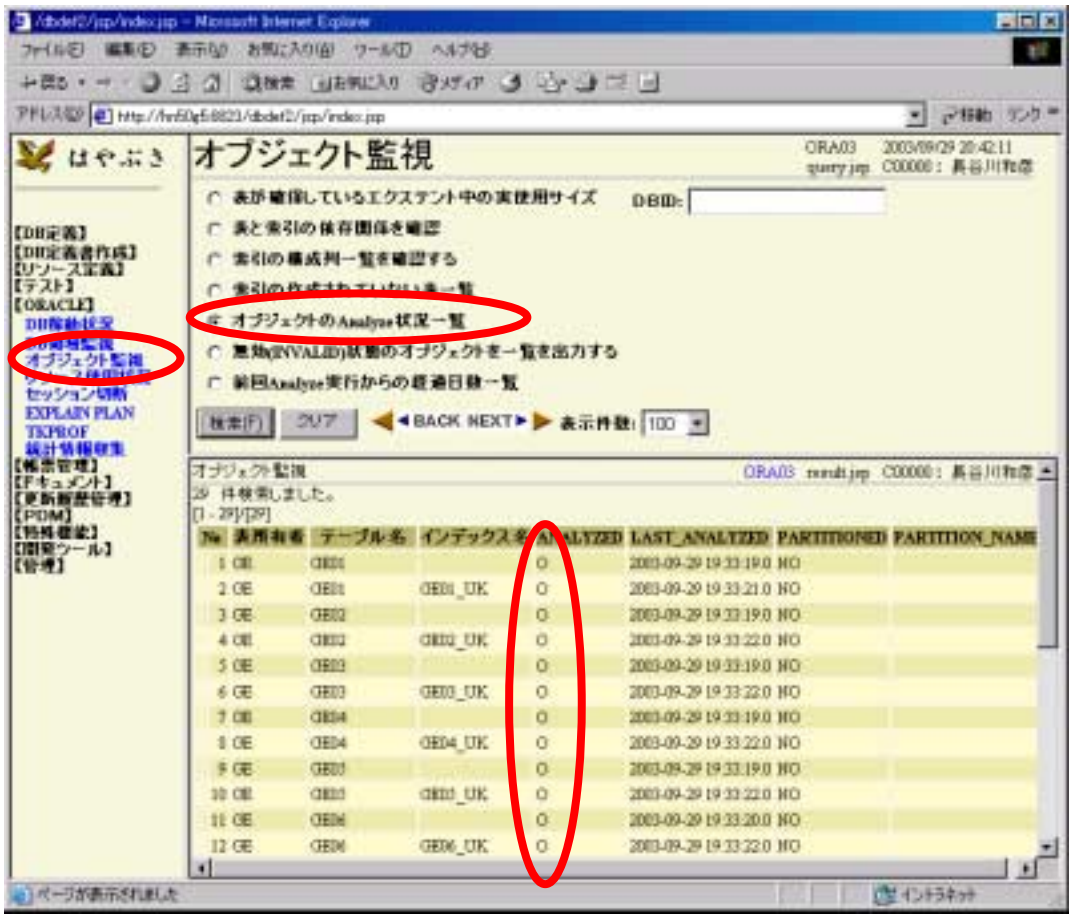
下記のサンプル画面では、『警告　SHARED_POOL_SIZE　を増やしましょう。』という警告が出ています。



3. オブジェクトの Analyze 状況一覧

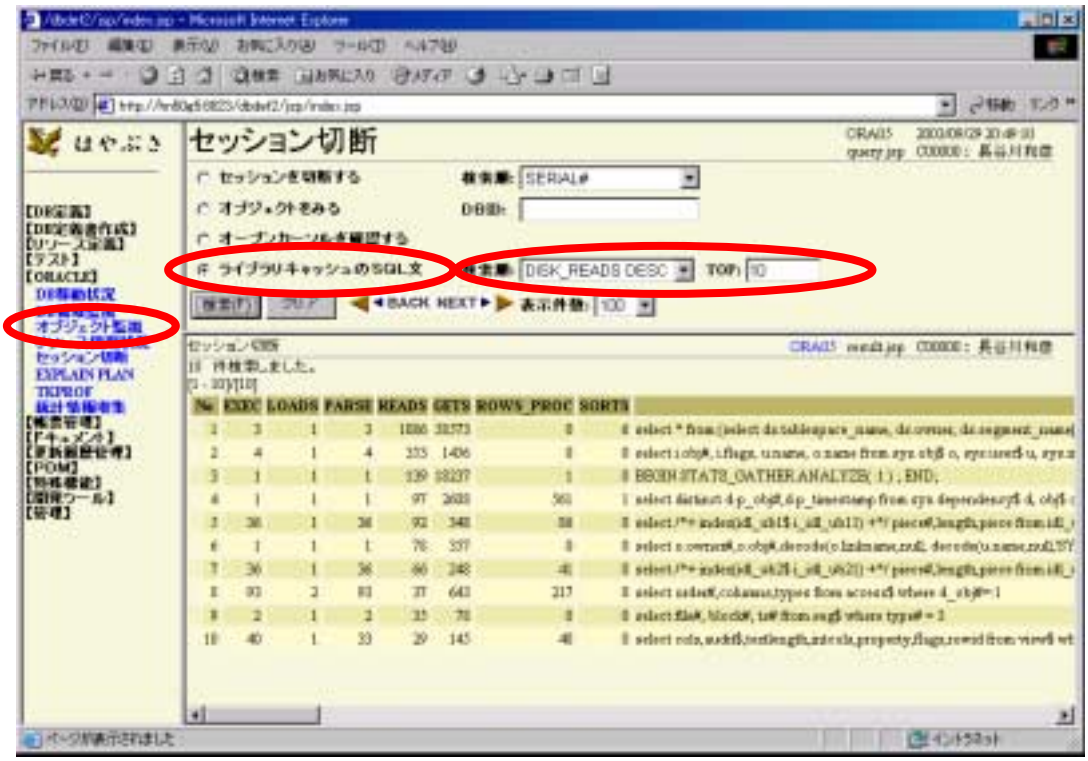
オブジェクトの Analyze 状況一覧では、表や索引について、Analyze されているかどうかを確認できます。初期化パラメーターの OPTIMIZER_MODE の初期値は、CHOOSE なので、テーブルが Analyze されていれば、コストベースの実行計画を作成します。

Analyze そのものの実行は、『統計情報収集』メニューで行います。



4. ライブラリキャッシュのSQL文

ライブラリキャッシュは、v$sqlarea ビューで確認することができます。これを解析することで、どのSQL文から、チューニングしていけばよいのかの、目安となる情報を見つけることができます。
このアプリケーションでは、ソート順と、トップN件の絞り込み表示ができます。

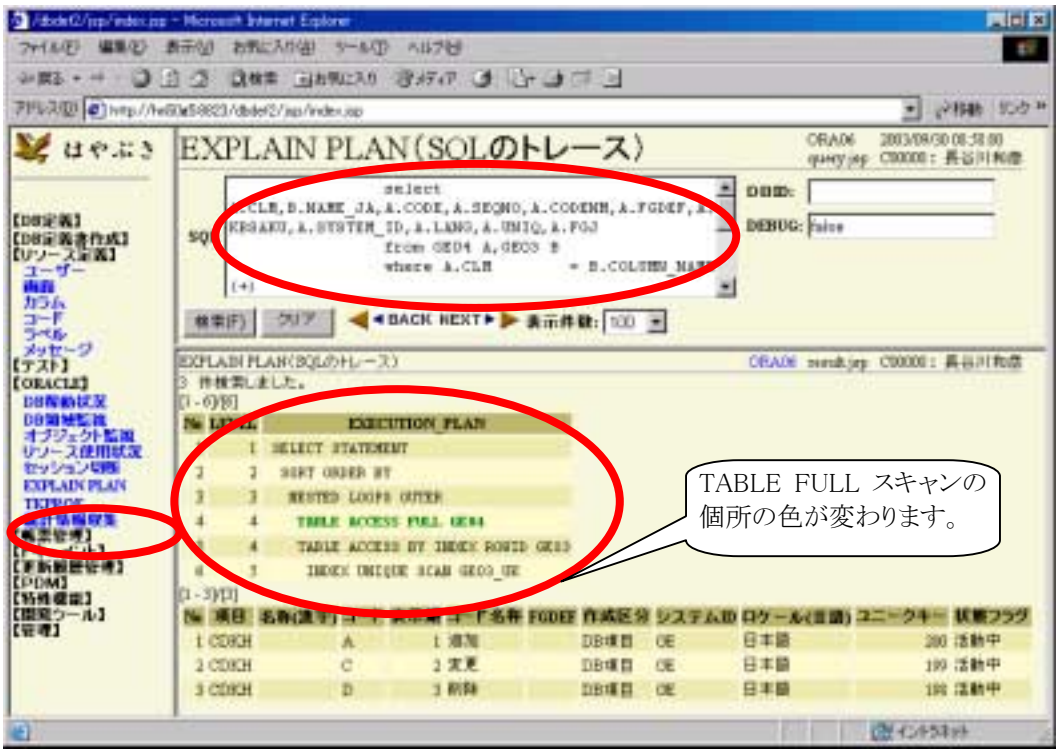


5. EXPLAIN PLAN

EXPLAIN PLAN 文は、SELECT 、UPDATE 、INSERT および DELETE 文について Oracle オプティマイザが選択した実行計画を表示します。文の実行計画とは、Oracle がその文を実行するために行う一連の処理です。

実行計画を求めたいSQL文をテキストエリアに書き込んで検索を行うと、実行計画とそのSQLの実行結果が表示されます。(通常の EXPLAIN PLAN は、SQLを実行しません。結果は、この画面で入力SQL文を実際に実行しているだけです。)
このSQL文には、最後に実行したSQL文が、自動的にセットされています。

EXPLAIN PLAN についての詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス リリース 2(9.2)』をご参照ください。



ヒント ≪EXPLAIN PLAN の実行≫
EXPLAIN PLAN を実行するには、まずは、プランテーブルを作成する必要があります。
スクリプトは、<ORACLE_HOME>/rdbms/admin ディレクトリに格納されています。
Windows の場合は、H:¥oracle¥ora92¥rdbms¥admin¥utlxplan.sql にあります。

6. TKPROF

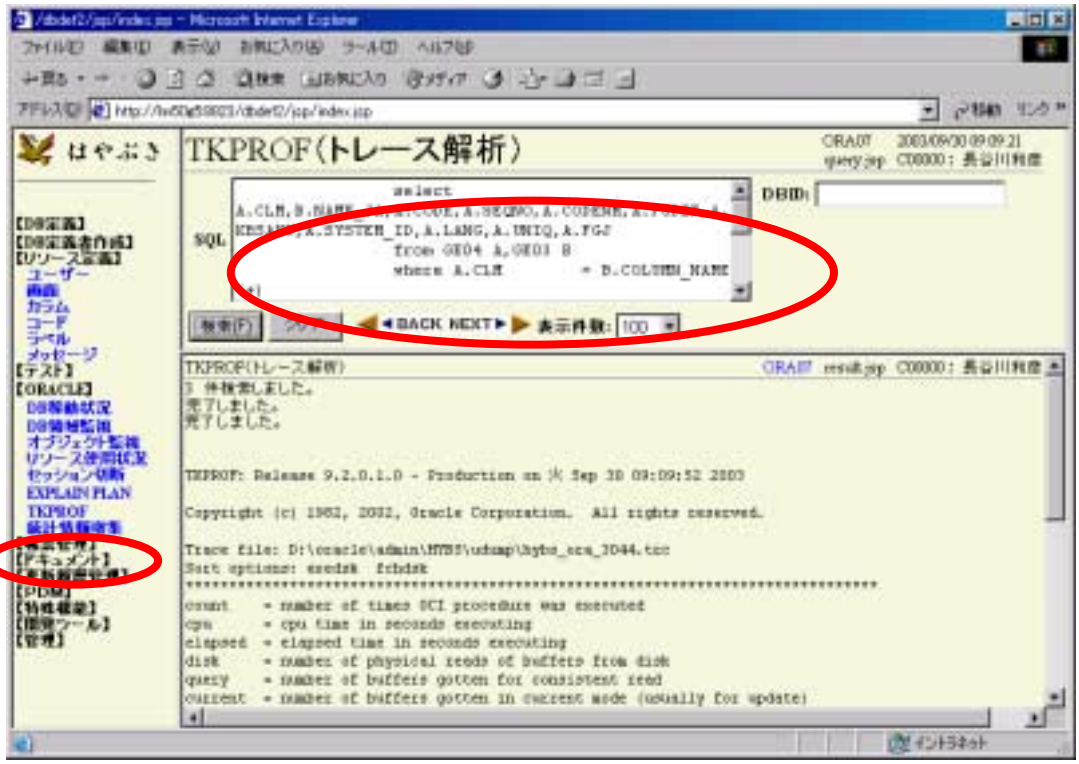
指定の SQL 文に対して、トレースを行う場合、query タグの trace 属性を true に設定します。ただし、エンジンではコネクションプーリングを行っているため、どのセッションでトレースされるか判らないため、トレースを利用する場合は、SystemResource.properties のコネクション数を1に設定しておく必要があります。

DEFAULT_DB_MINCOUNT = 1
DEFAULT_DB_MAXCOUNT = 1

この画面を利用すれば、個別の SQL について、トレースを行うことが可能です。

TKPROF は、トレース・ファイルの内容をフォーマットし判読可能なファイルとして出力できます。オプションとして、TKPROF は次のことも実行します。
TKPROF は、実行した各文を、消費したリソースおよびコールした回数、処理した行数とともにレポートします。この情報を使用すると、リソースを最も多く使用している文を簡単に検出できます。経験、または参考にできる基準をもとに、使用されたリソースが実行された作業に対して妥当であるかどうかを評価できます。

TKPROF についての詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス リリース 2(9.2)』をご参照ください。



ヒント ≪TKPROF の実行≫

TKPROF を実行するには、いくつかのステップを踏む必要があります。

1. SQL をトレースする。
   query タグで、trace="true" を設定すると、その実行セッションに対して、
   『ALTER SESSION SET SQL_TRACE=TRUE』を実行します。
   この SQL 文を実行し終わると、SQL_TRACE=FALSE でトレースを終了します。
   トレースを行うと、udump ディレクトリに、xxx.trc ファイルが作成されます。

2. tkprof プログラムを実行します。
   このプログラムは、xxx.trc ファイルを解析テキストファイルに変換します。
   トレースファイルの出力ディレクトリは、
   select value DIR from v$parameter where name = 'user_dump_dest'
   で、取得できます。
   ただし、トレースファイル名は、決まった名称でないため、バッチファイルより
   あいまい検索にて取得しています。

   ORA07/dunm_tkprof.bat ファイル
   for %%A in (%1¥*.trc) do tkprof %%A %1¥dump.txt （実際は、1 行）
   explain=ge/ge@hybs sort=(exedsk, fchdsk)

3. ORA07/dunm_tkprof.bat に、explain 属性として、DB 接続文字列を渡します。
   これは、Web サーバー側の、NET8 により接続可能である必要があります。

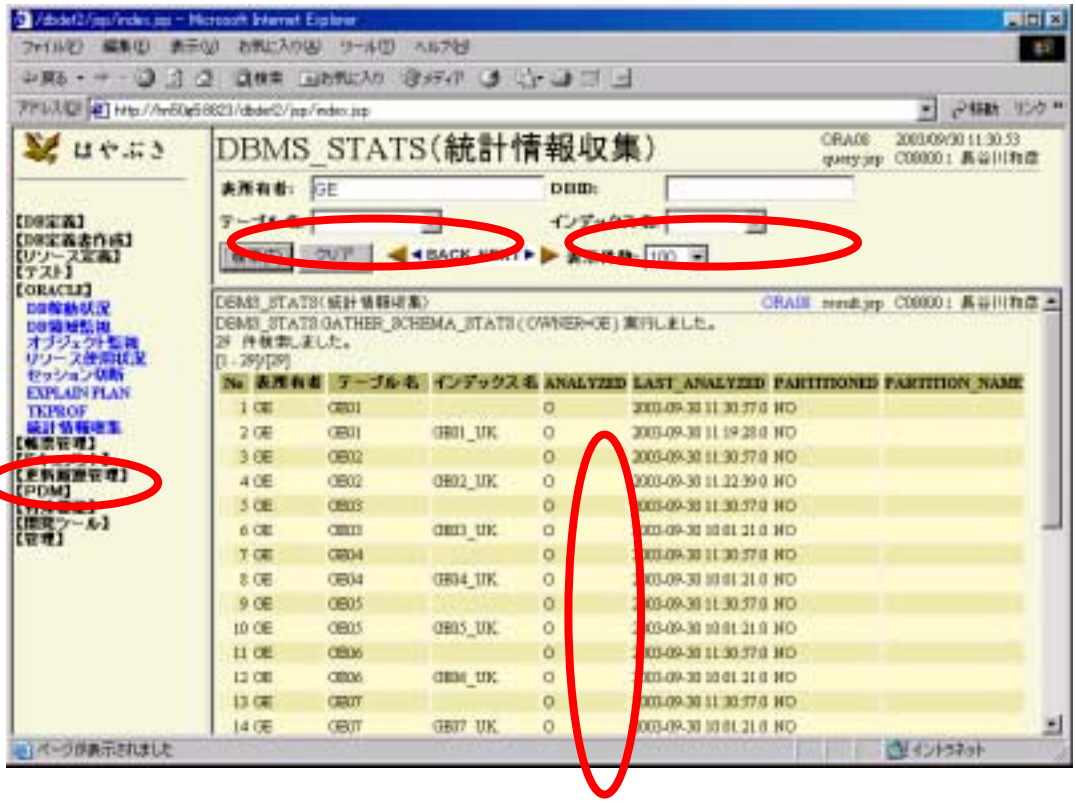
4. 出力の dump.txt ファイルを、表示します。

注意点としては、複数のトレースファイルが出力された場合は、このバッチでは、最後のファイルのみ、変換されます。
また、このトレースファイルは、解析後に、削除(別名で保存)されるため、SQL 文が実行される都度に、毎回トレースファイルを作成しています。

このアプリケーションでは、本格的な長期/複数の SQL に対して、トレースを行う使い方や、解析後のファイルの問題個所の洗い出しなど、今後の拡充が期待されています。

7. 統計情報収集

統計情報の収集を行います。ここでは、ANALYZE 文ではなく、DBMS_STATS パッケージを使用しています。テーブル名を指定した場合は、『DBMS_STATS.GATHER_TABLE_STATS』を、インデックス名を指定した場合は、『DBMS_STATS.GATHER_INDEX_STATS』を、どちらも指定しなかった場合は、『DBMS_STATS.GATHER_SCHEMA_STATS』を使用して、収集を行っています。
収集結果は、先に取り上げました、『オブジェクトの Analyze 状況一覧』検索画面と同じ SQL 文で表示させています。
ここでは、あくまで簡易的な機能のご紹介にとどめておきます。
詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス リリース 2(9.2)』、および、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス リリース 2(9.2)』をご参照願います。



第11章 ストレスツール（JMeter）

この章では、ストレスツールについて説明します。

Webアプリケーションに対して、実際にどれくらいのアクセスに耐えうるのか事前にテストしておくことは重要です。通常の範囲であれば、ハードウエアの増強で対応できますが、根本的に対応不可能な量になった場合、分散環境を考えることになります。その場合は、一気にハード、ソフトともにコストが跳ね上がります。（分散環境は、スケーラビリティー(scalability)だけでなく、アベイラビリティー(availability：可用性，有効性)のためにも構成されることがあります。）

もちろん、Web アプリケーションのボトルネックが、Web サーバーなのか、Web エンジンなのか、データベースなのか、各業務アプリケーションなのか、きちんと切り分けを行ってからでないと、対応方法が、異なってきます。

ここでは、Web アプリケーションに対して、負荷（ストレス）をかけるツールを説明いたします。

1．ストレスツール JMeterとは？

ORACLE 監視ツールには、以下の8つのメニューが用意されています。
ストレスツールには、有償/無償(※2)と色々あります。ここでは、無償のストレスツールのなかで、特に注目株の、JMeter を取り上げます。

※2 有名どころでは、Microsoft の『Web Application Stress Tool』があります。
これは、Web へのアクセスをそのまま記録して簡単に実行できます。
詳細は、下記 URL をご覧下さい。
http://support.microsoft.com/default.aspx?kbid=313559

JMeter は、jakarta プロジェクトで開発されている、無償・オープンソースの Java アプリケーションです。下記 URL で、JMeter のリンク 1.9.1 zip より入手します。

http://jakarta.apache.org/site/binindex.cgi

jakarta-jmeter-1.9.1.zip （2003年9月21日時点）

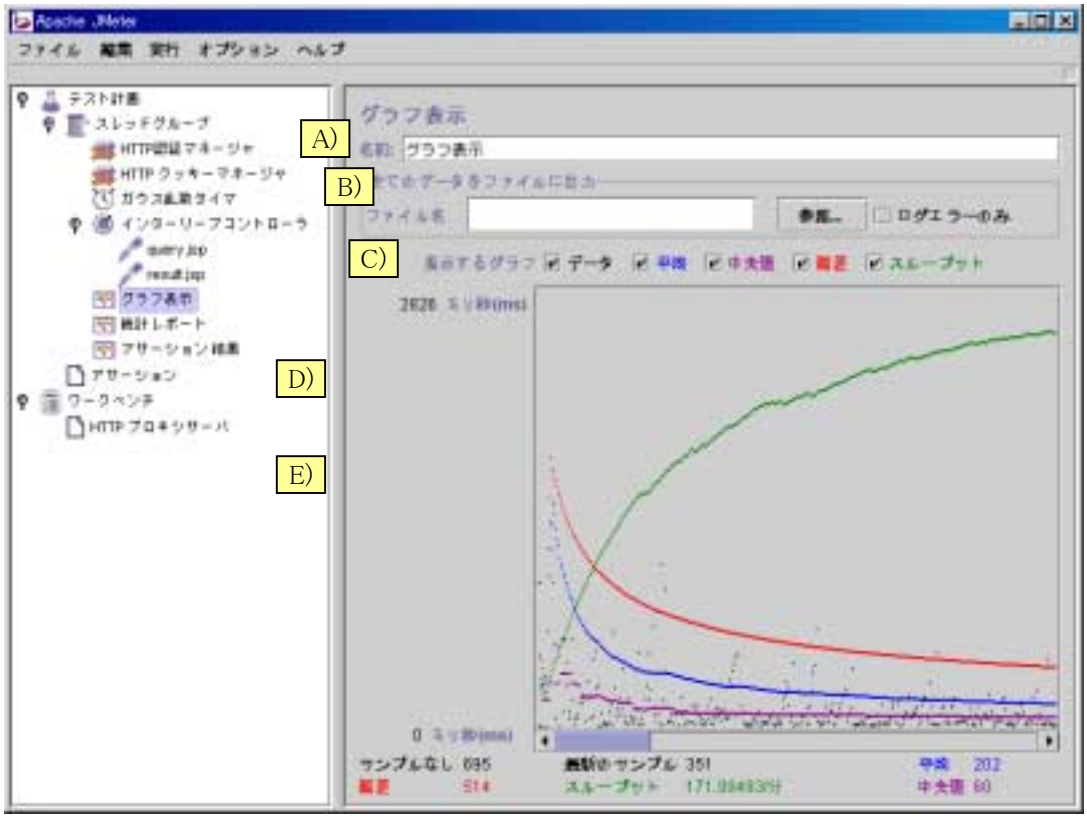
ZIP を解凍して、適当なディレクトリに置けばOKです。
bin/jmeter.bat をクリックして、きちんと日本語メニューで、画面が立ち上がれば、正常です。
日本語化には、jajakarta の多大なる貢献が活かされています。
（ http://www.jajakarta.org/ ）

2. JMeter の設定

実際に使用するまでに、各種負荷テストの条件を登録していく必要があります。また、実際に触ってみれば判りますが、リソースが日本語でも、設定方法は『コツ』のようなものがあり、解説記事がないと難しいと思います。

まずは、Web アプリケーションをテストするための設定だけはできるようにしてみましょう。

A).ログイン認証が必要。
B).ユーザーごとのセッションの追跡が必要。
C).指定の画面を、順次実行したい。
D).結果画面がエラーかどうか判断したい。
E).Web の操作をそのままテストしたい。



JMeterを立ち上げると、『テスト計画』と、『ワークベンチ』があります。この、『テスト計画』に、右ボタンでメニューを表示し、必要な機能を追加していきます。
まず、はじめに、テスト計画に、スレッドグループを追加します。ここで、同時接続数やループ回数を指定します。Ramp－Up期間とは、その時間間隔で、スレッドを順番に立ち上げていきます。
この、スレッドグループに、『ロジックコントローラ』、『リスナー』、『サンプラー』、『タイマ』、『設定エレメント』などを選んでいきます。

A).ログイン認証が必要。
⇒ 設定エレメントの**『HTTP 認証マネージャ』**
B).ユーザーごとのセッションの追跡が必要。
⇒ 設定エレメントの**『HTTP クッキーマネージャ』**
C).指定の画面を、順次実行したい。
⇒ ロジックコントローラの**『インタリーブコントローラ』**

これら以外に、『タイマ』で、スレッドの実行タイミングを調整したり、『リスナー』で、『統計レポート』や、『グラフ表示』 など、ビジュアル的に結果を確認することができます。
アサーションとは、実行結果に対して、必要な文字列が含まれている、または、含まれていないなど、判定できます。これにより、結果がエラーかどうか判断することができます。

D).結果画面がエラーかどうか判断したい。
⇒ テスト計画で、『アサーション』と、
スレッドグループのリスナーの**『アサーション結果』**

また、Webで操作したURLそのものを HTTP リクエストとして取得する機能に、『HTTP プロキシサーバ』があります。つまり、JMeterをプロキシサーバーに設定することで、アクセスURLをそのまま取り込むことができます。

E).Web の操作をそのままテストしたい。
⇒ ワークベンチで、『non-Test エレメント』で**『HTTP プロキシサーバ』**

これら以外に、JMeterサーバーとして、RMIを使って遠隔操作により大量の負荷テストを分散サーバー上から行うことも可能です。しかし、テスト結果の解析などは、まだ十分ではありませんが、オープンソースの強みを見せて、今後の充実に期待したいと思います。

コラム

某メーカー殿による、Tomcat、OC4J、9iAS のパフォーマンス測定結果です。
測定は、『Web Application Stress Tool』を用いて行われています。
詳細は取り上げませんが、スループット、応答時間でいうと、大差はでませんでした。
この結果から言うと、100同時ユーザーなら問題なく動作しています。
Web サーバーを2CPU にするなどにより、200同時ユーザーでも、たぶん問題ないでしょう。

Oracle 9i Application Server と Tomcat に関する検証報告書

Tomcat4.1.24
Oracle9iAS Containers for J2EE(OC4J)9.0.3 (以下 OC4J)
Oracle9iAS 9.0.3 (以下 9iAS)

Database
Sun Enterprize 4500 OS:Solaris2.6
CPU:UltraSPARC-Ⅱ(366MHz) 8way Memory:2GB
Oracle9i Database Release2(9.2.0.1)

Application Server(Tomca,OC4J,9iAS)
Dell OptiPlex GX260 OS:Windows 2000
CPU:Pentium4(2.4G) 1way Memory 1.5GB
Tomcat 4.1.24 / J2SE1.4
Oracle9iAS Containers for J2EE(OC4J)9.0.3 / J2SE1.4
Oracle9iAS 9.0.3 / J2SE1.4

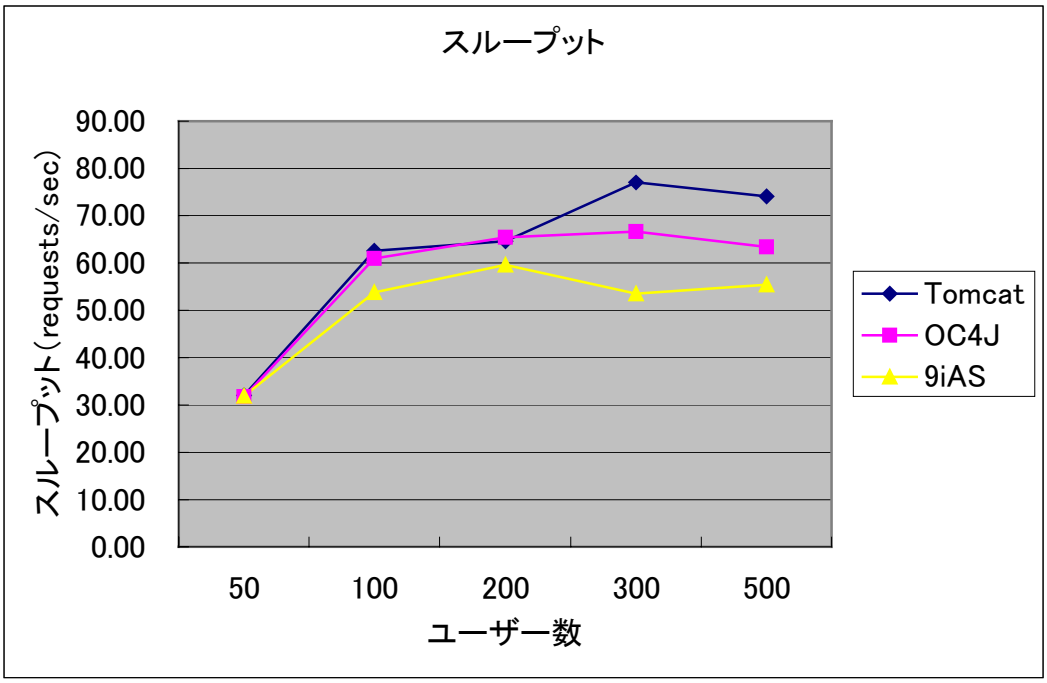
チューニング(Tomcat)

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
                    port="8823" minProcessors="5" maxProcessors="200"
                    enableLookups="false" redirectPort="8443"
                    acceptCount="200" debug="0" connectionTimeout="20000"
                    useURIValidationHack="false" disableUploadTimeout="true" />
```

※ acceptCount="200" に設定したのは、まずかったかもしれません。200ユーザーオーバーから、ソケットエラーなるリクエストが増えてきました。(OC4J も Tomcat と同様のエラーを示す。) 9iAS では、ソケットエラーは発生していないので、(EJB エンジン部は、OC4J と同じ。フロントエンドに Apache を使っているだけ。)アクセス関係の設定により、防げると考えられます。

スループット

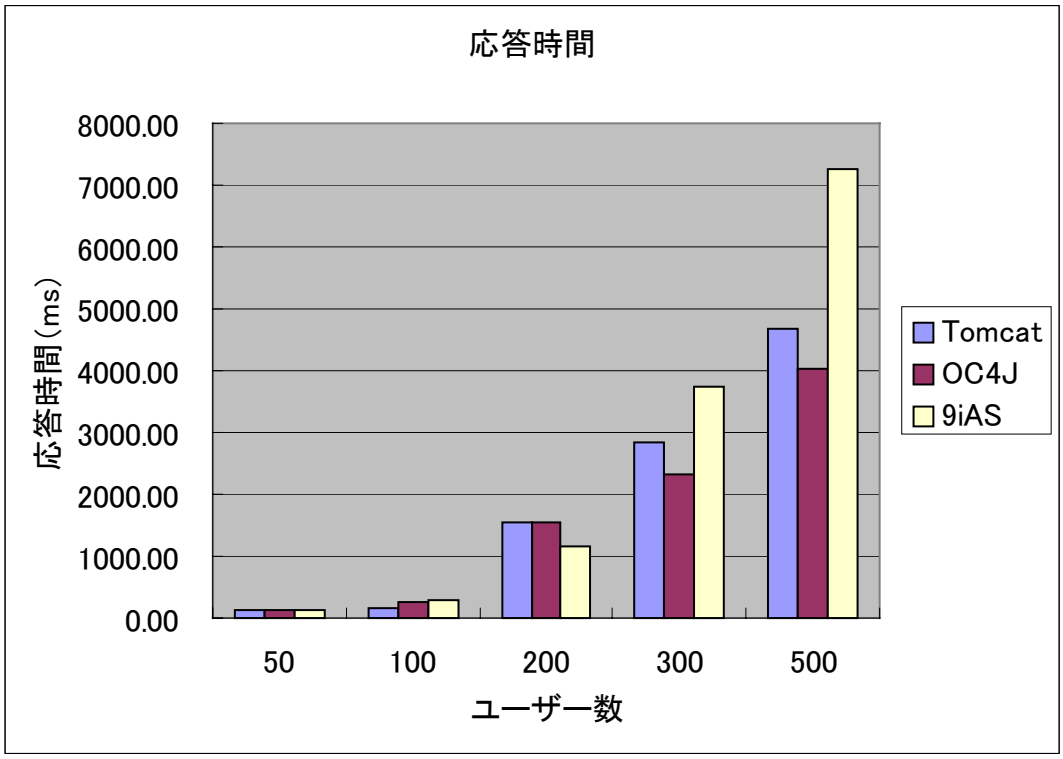
	ユーザー数				
	50	100	200	300	500
Tomcat	32.04	62.61	64.60	76.96	74.09
OC4J	31.81	61.00	65.38	66.63	63.39
9iAS	31.95	53.73	59.62	53.51	55.42



50ユーザーから100ユーザーに対して、スループットが倍になっています。つまり、速度を落とさず100ユーザーまで、応答していることを示しています。
200ユーザーでも、100ユーザーと同じだけのスループットを確保しているということは、応答時間が延びてはいるが、大体このあたりがスループットの限界であることが判ります。
300ユーザーでは、9iAS のスループットが落ち、Tomcat は、少し伸びています。
エラー等の解析をしないと不明ですが、300ユーザーでは、すでに限界を超えていると判断できます。

応答時間

	ユーザー数				
	50	100	200	300	500
Tomcat	116.03	166.24	1551.34	2825.06	4663.25
OC4J	130.99	254.79	1532.78	2333.96	4034.69
9iAS	133.12	284.81	1159.69	3753.99	7245.18



100ユーザーまででであれば、ほとんど応答時間が増えません。これは、先のスループットが倍増したように、応答時間を落とさずレスポンスを返していることが判ります。
200ユーザーで、応答時間が延びてきます。
ただし、ページエラー（ソケットエラー）が、300ユーザーあたりから増えています。OC4J の方が、Tomcat よりもエラー件数が多いため、（つまり、リクエストを処理していない）応答時間が短くなっていると考えられます。9iAS は、エラーを起こしていないため、応答時間が伸びていると考えられます。
エラーについては、Web サーバーの設定で、対応できると思います。

第 V 部 アプリケーション負荷分散

『我々は人生の大半の時間を仕事に使っている。
仕事が楽しければ、誰でも熱心に働く。
仕事が楽しくなるのなら、できることは何でもするべきだと思う。
私の一番大切な仕事は、仕事を楽しくすることだ』

アンドリュー・グローブ（インテル創業者）

ここでは、アプリケーションの負荷分散について説明します。
構成は次のとおりです。

第 12 章　Apache と Tomcat との連携
負荷分散に Apache の JK2 モジュールを使用する前段階として、Apache と Tomcat の連携を行います。

第 13 章　Apache と Tomcat による負荷分散
JK2 による負荷分散機能（ロードバランサー）の設定方法を解説します。

第12章 Apache と Tomcat との連携

ここでは、負荷分散に Apache の JK2 モジュールを使用する前段階として、Apache と Tomcat の連携を行います。

1． Tomcat-Standalone サービス

Web エンジンを使用したアプリケーションでは、原則、Web サーバーは、Tomcat そのものを使用しています。理由は次のとおりです。

- **サーバーのメンテナンスが簡単（インストール、設定、運用管理）**
- **Apache 等の Web サーバーと組み合わせても、レスポンスにそれほど大きな差がない。**
- **ほとんどの業務アプリケーションは、JSP のみで動作している。（ほとんど動的画面）**
- **ほとんどの業務アプリケーションは、LAN(社内ネットワーク)のみで運用されている。**
- **業務アプリケーションの静的ドキュメントは、インターネットのそれよりはるかに大きい。**

例えば、静的ドキュメントであれば、Tomcat より Apache の方が早いという話がありますが、業務で使用するドキュメントの場合、CAD ファイルや図面、仕様書など、一般のインターネット上のファイルより、はるかに大きなファイルがほとんどで、運用としては、図面サーバー等へのリンクを JSP で作成するケースが、多いと思われます。
また、そのようなファイルは、大多数が同一ファイルを取得する（つまり、キャッシュが有効）ではなく、その設計担当者や業務担当者が、個別に使用するため、厳密な効果が期待できない状態で、複雑なサーバー形態をとることは、リスクが増えると考えています。

現に、JDK1.4.2 と、Tomcat4.1.27 の組み合わせでは、一般的な Windows サーバー（※）で、100同時ユーザー程度であれば、問題ありません。
※ Pentium4(2.4G) 1way Memory 1.5GB （コラム参照）

Apache と連携させなければいけない理由として、以下の項目が挙げられると思います。

- 外部（インターネット）からの接続等で、セキュリティをきちんと取りたい。
- 複数の Tomcat に対して、負荷分散による拡張性(scalability)を高めたい。
- 複数の Tomcat を用いて、負荷分散による可用性（availability）を高めたい。

あくまで、個人的な見解と断りを入れさせていただきますが、実際に、Web エンジンでターゲットにしている業務分野は、製造業（生産管理等）が主流のため、外部からのアクセスや、負荷分散による拡張性・可用性の確保の必要性は、あまり多くありません。

つまり、無意味に、複雑なシステムを構築することは、得策ではありませんので、十分理解した上で、Apache との連携をご判断いただきたいと思います。

2. Apache と Tomcat の連携

まずは、『Apache』について、入手、インストール、設定までを解説してみたいと思います。

①Apacheの入手、インストール

http://httpd.apache.org/

Download ｜ from a mirror を選んで、最寄のミラーサイトからダウンロードしてください。ここでは、binaries の win32 より、 **apache_2.0.47-win32-x86-no_ssl.exe** を選びます。(2003年 8 月17日時点)
入手したEXEファイルをダブルクリックして、インストールします。

②設定

インストールフォルダ下の Apache2¥conf¥httpd.conf ファイルを、編集しておきます。

ドキュメントベースを変えます。

```
DocumentRoot "D:/InetPub/wwwroot"
<Directory "D:/InetPub/wwwroot">
```

ドキュメントのデフォルトエンコード(※1)を、HTML ファイルにあわせて、設定しておきます。その他の設定箇所はマニュアルや市販のApache本を参考に各自で色々試してみてください。

```
#AddDefaultCharset ISO-8859-1
    AddDefaultCharset shift-jis
```

※1 AddDefaultCharset の設定があると、HTML の META タグに応じたエンコードを返しません。よって、デフォルトでは漢字が文字化けするため、コメントアウトを勧める記事等がありますが、クロスサイトスクリプト等のセキュリティ上の問題が発生することがあるため、きちんとファイルに応じたエンコードを指定しておいてください。

参照URL:
http://httpd.apache.org/info/css-security/apache_specific.html
http://www.cert.org/tech_tips/malicious_code_mitigation.html

③実行確認

指定のURL(例えば、http://localhost/index.html)で画面が表示されれば、OK です。

④ Tomcat との連携

ここからが、本命です。
Tomcat4．1 の Coyote JK2 コネクタ(以下 JK2)を用いて、Apache2 との接続を行ってみます。 まずは、mod_jk2 の dll を入手します。

http://jakarta.apache.org/builds/jakarta-tomcat-connectors/jk2/release/v2.0.2/bin/win32/

mod_jk2-2.0.43.dll （2003年9月18日時点）

1)．mod_jk2-xxx.dll を、<<Apache2>>/modules にコピー
2)．<<Apache2>>/conf/httpd.conf に、モジュール追加

```
LoadModule jk2_module modules/mod_jk2-2.0.43.dll
```

3)．<<Apache2>>/conf/workers2.properties
ファイルを追加。(超簡易設定)

```
[logger.file:0]
level=ERROR
file=${serverRoot}/logs/jk2.log

[channel.socket:localhost:8009]
info=Ajp13 forwarding over socket

[uri:/training/*]
info=Training Web Apprication

# [status:]
# [uri:/jkstatus/*]
# group=status

[shm:]
disabled=1
```

この URL パターンを、Tomcat に転送します。

[status:]と[uri:/jkstatus/*]は設定状況を確認する場合に使います。
本番環境ではコメントしておきます。

[shm:]は、Shared memory の設定です。使用しませんので、disabled に設定します。

4)．<<Tomcat4.1>>/conf/jk2.properties
※ デフォルトで良いなら、さわる必要ありません。
5)．<<Tomcat4.1>>/conf/server.xml
CoyoteConnector の個所のコメントをはずします。
※ Ajp13Connector と、間違えないように。

```
<Connector className=
    "org.apache.coyote.tomcat4.CoyoteConnector"
    port="8009" minProcessors="5" maxProcessors="75"
    ・・・・・・
```

6)． Tomcat ⇒ Apache の順で起動します。
7)． http://localhost/training/jsp/index.jsp で
画面が表示できれば、OK です。

ヒント

ここまでの設定は、単に jsp へのアクセスをすべて Tomcat に転送しているだけです。Tomcat 側の静的ファイルを Apache で処理させる場合は、servlet と jsp 拡張子ファイルのみを転送し、Apache のドキュメントベースと Tomcat のドキュメントベースを合わせておく必要があります。

第13章 Apache と Tomcat による負荷分散

ここでは、JK2 による負荷分散機能（ロードバランサー）の設定方法を解説します。

設定が必要なのは、Apache の workers2.properties と、Tocat の server.xml ファイルです。

1）. <<Apache2>>/conf/workers2.properties
色付き太字（下線）の部分が、今回の追加分です。

```
[logger.file:0]
・・・・・<<省略>>・・・・・

# load balancing Group
[lb:lb]

# Host hn50g5
[channel.socket:hn50g5:8009]
info=Server hn50g5
host=hn50g5
tomcatId=hn50g5
lb_factor=1
## port=8009
## group=lb

# Host hn51d4
[channel.socket:hn51d4:8009]
info=Server hn51d4
host=hn51d4
tomcatId=hn51d4
lb_factor=2
## port=8009
## group=lb

[uri:/training/*]
info=Training Web Apprication
  以下、変更なし。
```

ロードバランスグループの設定です

ホスト hn50g5 の設定です。tomcatId は、セッショントラッキング用の識別文字列です。

ホスト hn51d4 の設定です。

2). <<Tomcat4.1>>/conf/server.xml
Engine 要素に、jvmRoute 属性を記述します。
これは、workers2.properties の tomcatId=hn51d4 で記述した値と一致させます。
このキーは、セッション ID 末尾に付加され、セッショントラッキングを実現します。

ホスト hn50g5 の server.xml

```
<Engine name="Standalone" defaultHost="localhost"
        debug="0"  jvmRoute="hn50g5" >
```

ホスト hn51d4 の server.xml

```
<Engine name="Standalone" defaultHost="localhost"
        debug="0"  jvmRoute="hn51d4" >
```

3). Tomcat ⇒ Apache の順で起動します。
4). http://localhost/training/jsp/index.jsp でアクセスし、
lb_factor で指定された負荷係数でアクセスするサーバーが変われば、OK です。

ヒント ≪lb_factor での負荷のかけ方≫
lb_factor での設定値は、実際の負荷のかかり方と、逆になります。
例えば、hn50g5 の ib_factor=1 , hn51d4 の ib_factor=2 とすると、
hn50g5:hn51d4 = 2:1 の比率で、アクセスされます。
この負荷のかけ方は、ラウンドロビン方式です。ラウンドロビン方式とは、アクセスされる都度に、次々とサーバーを移っていく方式で、サーバー(Tomcat)の負荷とは関係なく割り振られます。もちろん、片方が落ちた状態では、もう片方に集中的に振り分けられ、復旧した段階から、再び順番に割り振られます。
割り振り方は、先の lb_factor の設定値に応じて振り分けられます。

※ ただ、こちらで lb_factor にともに、1を設定したときに、うまく振り分けられませんでした。ずっと、同じ方のサーバーにばかり振られていました。そのサーバーを停止させると、初めてもうひとつのサーバーに振られました。
この原因は、不明です。
1:2 にすると、きちんとその割合で振られました。

コラム ≪Web エンジンのロードバランサー≫

Web エンジンに、proxy というアプリケーションがあります。
これは、簡易的な負荷分散処理をラウンロビン方式で行うことができます。
内容的には、Tomcat そのものを複数立ち上げておき、proxy への最初のアクセスで順次それらの Tomcat に sendRedirect するというものです。
よく見かける、リンクが変更になった場合に、自動的に転送されるのと同じ原理です。
この proxy の利点は、とにかく簡単であることです。また、転送されるため、それ以降のアクセスは、直接クライアントと tomcat サーバーとで行われますので、セッショントラッキングは必要ありません。
欠点としては、転送先の Tomcat が活きているかどうかの確認をせずに転送しているため、転送先でエラーになる可能性があります。また、負荷係数は、現在のところ、かけていません。(負荷係数の実装は比較的簡単ですけれども・・・)

ビジネスとして販売する場合に、あまりにもダサい方法ではバカにされたり、相手にされない場合がありますが、実際の運用となると、実のところ、この程度の仕掛けで十分実用的です。

その技術が本当に必要なのか、簡易的な方法で代用できないか、検討するだけの価値はあると思います。

索引