Oracle Servlet Engine
補足ドキュメント

Creation Date:	Nov 09, 2000
Last Update:	Nov 20, 2000
Version:	1.0

ORACLE®
SOFTWARE POWERS THE INTERNET™

1.概要と基礎知識

Oracle Servlet Engine (OSE)は、Oracle8*i* 内で動作するスケーラブルなサーブレット実行環境です。また、OSE はサーブレットの実行環境であると同時に Web サーバーとしての機能も提供します。各ブラウザ・セッションに、一つの仮想的な JavaVM が割り当てられるため、OSE は優れたスケーラビリティを提供します。また、データベースへのアクセスは JDBC サーバー側内部ドライバを使用するため高速です。

サーブレットのクラスは、loadjava ユーティリティを使用して Oracle8*i* に格納し、セッション・シェルを使用して公開されます。OSE は、HTTP のリクエストを受け、公開されているサーブレットをインスタンス化し、メソッドを呼び出します。

1-1. OSE の内部構成

ここでは、まず OSE の内部構成を解説します。OSE は、

- Web サービス
- エンドポイント
- Web ドメイン
- サーブレット・コンテキスト
- サーブレット

から構成されます。この構成を表したものが、図 1-1 になります。

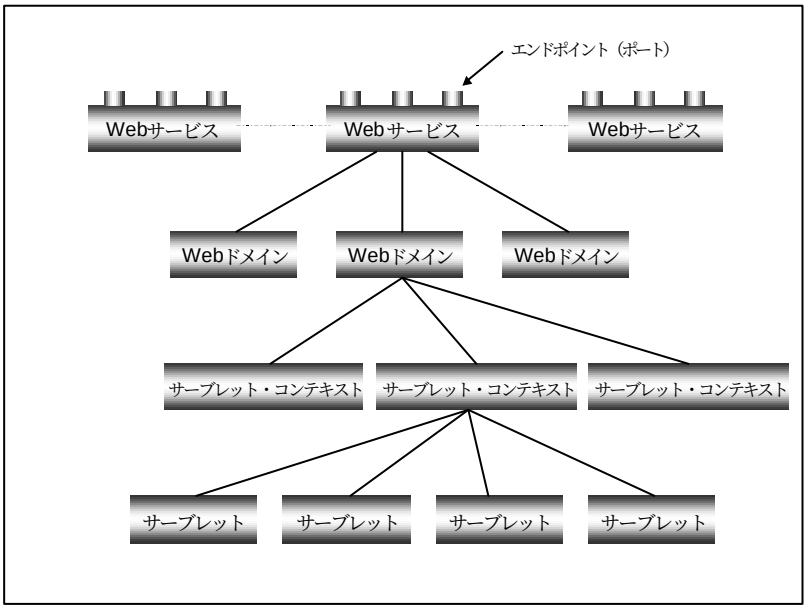


図 1-1. OSE の内部構成

1-1-1. Webサービス

OSEでは、複数のWebサービスを提供することができます。Webサービスは、ホスト名とポート番号で決まるWebサーバーのインスタンスのようなものです。1つのWebサービスには、1つ、もしくは複数のWebドメインが含まれます。OSEでは、シングル・ドメインWebサービスとマルチ・ドメインWebサービスという2種類のWebサービスの設定をすることができます。

シングル・ドメインWebサービス

Webサービスは1つのWebドメインで構成されます。ほとんどの場合は、この設定で十分です。

マルチ・ドメインWebサービス

複数のホスト名または複数のIPアドレスを使用する場合、Webサービスは複数のWebドメインを持ちます。マルチ・ドメインWebサービスでは、ホスト名かIPアドレスのどちらか、もしくは両方を使用して、リクエストを正しいWebドメインにルーティングします。

1-1-2. エンドポイント

Webサービスは、いくつかのエンドポイントを持ちます。エンドポイントとは、接続を受け付ける入り口（例えばポート）のことです。

1-1-3. Webドメイン

ホストは、複数のホスト名および複数のIPアドレスを持つことがあります。このとき、それぞれのホスト名、IPアドレスで決定される仮想ホストがWebドメインです。

・シングル・ドメインWebサービスの場合

Webサービスは単一のWebドメインを持ちます。つまり、どのようなホスト名(DNS名)、IPアドレスでアクセスしても同じWebドメインが使用されます。

・マルチ・ドメインWebサービスの場合

-複数のホスト名を持つ場合

1つのWebサービスが複数のDNS名を持つ場合、それぞれを1つのWebドメインに対応付けします。

-複数のIPアドレスを持つ場合

1つのWebサービスに複数のネットワーク・インタフェースが存在する場合、各IPアドレスに対し、それぞれWebドメインを対応付けします。

Web ドメインは、1 つ以上のサーブレット・コンテキストで構成されます。

1-1-4. サーブレット・コンテキスト

1 つのサーブレット・コンテキストは、1 つの Web アプリケーションと考えることができます。サーブレット・コンテキストは、サーブレットや Init パラメータ、JSP、そして静的コンテンツに対するポインタで構成されます。

サーブレット・コンテキストは、URL 上では 1 つのコンテキスト・パス(仮想パス)にマッピングされます。コンテキスト・パスに関しては、「1.4.1 URL の構成」で解説します。

1-2. OSE の設定を行うツール：sess_sh

OSE の管理には、主にセッション・シェル（sess_sh）を使用します。セッション・シェルには、ディレクトリの移動や管理のために、cd、pwd、ls、mkdir、rm といった標準的な UNIX のシェルライクなコマンドが用意されています。

また、このような UNIX のシェルライクなコマンドの他にも、JNDI 名前空間（後述）を管理するための特別なコマンドも用意されています。次に、そのうち使用する頻度の高いものを解説します。

getproperties

オブジェクトのプロパティの一覧を表示します。

```
$ getproperties <オブジェクト名>
```

例えば、次のようにします。

```
$ getproperties config
--group--=environment
--group--=contexts
/red=planet
--group--=mime                          … グループ名
java=text/plain
html=text/html
htm=text/html    … グループ・エントリ
body=text/html
xml=application/xml
txt=text/plain
rtx=text/richtext                       } プロパティ・グループ
tsv=text/tab-separated-values
etx=text/x-setext
ps=application/x-postscript
csh=application/x-csh
sh=application/x-sh
tcl=application/x-tcl
… 省略 …
```

setproperties

オブジェクトのプロパティを設定します。

```
$ setproperties <オブジェクト名> “<プロパティ名 1>=<値 1> <プロパティ名 2>=<値 2> …”
```

getgroup

「--group--=environment」のように、オブジェクトにプロパティ・グループがある場合、指定されたグループのプロパティの一覧を表示します。

```
$ getgroup <オブジェクト名> <グループ名>
```

addgroupentry

すでにあるプロパティ・グループにプロパティを 1 つ追加します。

```
$ addgroupentry <オブジェクト名> <グループ名> <プロパティ名> <値>
```

setgroup

プロパティ・グループを作成します。

```
$ setgroup <オブジェクト名> <グループ名> “<プロパティ名 1>=<値 1> <プロパティ名 2>=<値 2> …”
```

removegroupentry

プロパティ・グループから指定のプロパティを削除します。

```
$ removegroupentry <オブジェクト名> <グループ名> <プロパティ名>
```

スクリプトの実行

セッション･シェルコマンドを記述した OS **上のスクリプト・ファイルを実行します。例えば** test.ssh **というセッション･シェルのスクリプトを実行するためには**

```
$ @test.ssh
```

と入力します。

各コマンドの詳細は、「Oracle8*i* Java Tools **リファレンス」マニュアルを参照してください。**

1-3. OSE の JNDI 名前空間

JNDI(Java Naming and Directory Interface)**名前空間には、**OSE **や** CORBA**、**EJB **といった異なるタイプの** Java **のサービスの情報やコンテンツが格納されています。**OSE **の管理では、主にセッション･シェルを使用して、**JNDI **名前空間内の検索や操作をします。この名前空間は、「ディレクトリ」と「オブジェクト」という** 2 **種類のエントリで構成されています(図** 1-2)**。また、各** JNDI **のエントリには、**READ**、**WRITE**、**EXECUTE **という** 3 **種類のパーミッションがあり、セッション・シェル上で** chmod **コマンドを使用して、各データベースユーザーにこれらのパーミッションを与えることができます。**

OSE **で主に使用するディレクトリとオブジェクトは図** 1-2 **のようになっています。**

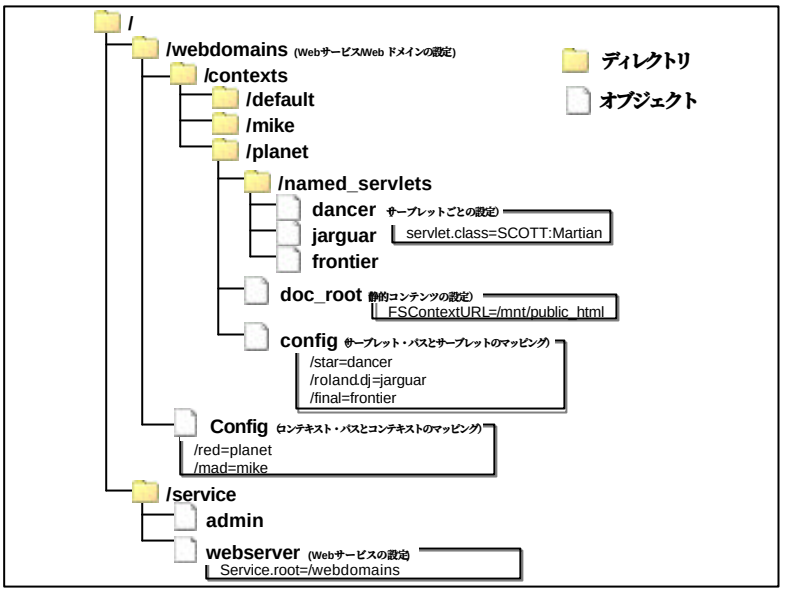


図 1-2. OSE での JNDI 名前空間 (シングル・ドメインの例)

リクエストがあると、OSE はこの JNDI 名前空間からリクエストの URL に該当するサーブレットを

Web サービス > Web ドメイン > サーブレット・コンテキスト > サーブレット

の順で検索します。ここでは、webserver というシングル・ドメイン Web サービスがあると想定して、サーブレットをどのように検索していくかを解説します。

1-3-1. OSE 全体の設定情報

OSE 全体の設定は、/service ディレクトリ内にあります。/service ディレクトリには、現在 OSE にある Web サービスの設定情報を持つオブジェクトがあります。図 1-2 では、初期状態からある admin という Web サービスと、新たに作成した webserver という（シングル・ドメイン）Web サービスの設定情報を持つオブジェクトがあります。

1-3-2. Web サービスの設定情報

webserver というオブジェクトのプロパティがリスト 1-1 になります。

```
$ cd service
$ getproperties webserver
--group--=service
service.name=Service webserver
```

```
service.description=Aurora HTTP Servlet Engine
service.version=1.0
service.vendor=Oracle Corp.
service.globalTimeout=60
service.root=/webdomains
service.presentation=http://webserver
service.error.log=servicelogs/error
service.event.log=servicelogs/event
--group--=endpoint
endpoint.class=SYS:oracle.aurora.mts.ServiceEndpoint
endpoint.name=endpnet8
endpoint.endpnet8.interface=*
endpoint.endpnet8.port=0
endpoint.endpnet8.backlog=50
endpoint.endpnet8.min.threads=3
endpoint.endpnet8.max.threads=5
endpoint.endpnet8.timeout=300
--group--=environment
--group--=contexts
--group--=mime
java=text/plain
html=text/html
htm=text/html
body=text/html
xml=application/xml
txt=text/plain
rtx=text/richtext
… 省略 …
```

リスト 1-1. webserver オブジェクトのプロパティ

このオブジェクトには、service や endpoint、mime といったプロパティ・グループがあります。例えば、service グループには、この Web サービスの環境の情報をプロパティとして持ち、endpoint グループには、この Web サービスのエンドポイントの設定情報が格納されています。service.root というプロパティで、Web サービスのルートディレクトリが確認できます。

1-3-3. Web ドメインの設定情報

Web ドメインのルートディレクトリ（/webdomains）には、この Web ドメインのコンテキストを格納している contexts ディレクトリと、この Web ドメインの設定情報を格納する config オブジェクトなどが含まれます。この config オブジェクトには、Web ドメインのサーブレット・コンテキストの情報や mime タイプの設定などが格納されています。

シングル・ドメイン Web サービスの場合は、Web サービスのルートディレクトリと Web ドメインのルートディレクトリは一致します。今回の例はシングル・ドメイン Web サービスの例なので、/webdomains ディレクトリは Web サービスのルートであり、Web ドメインのルートでもあります。

今回の例では Web サービスのルートは/webdomains という名前のディレクトリですが、シングル・ドメイン Web サービスであるため、ここには 1 つの Web ドメインの情報しか格納されていません。マルチ・ドメイン Web サービスの設定に関しては「5-4 マルチ・ドメイン Web サービスの設定」を参照してください。

```
$ cd webdomains
$ getgroup config contexts
/red=planet
/mad=mike
```

リスト 1-2. config オブジェクトの contexts グループのプロパティ

リスト 1-2 にある contexts というプロパティ・グループには、URL のコンテキスト・パスと JNDI 名前空間内でのコンテキスト名とのマッピングの情報が格納されています。リスト 1-2 では、/red というコンテキスト・パスが planet というコンテキストにマップされていることがわかります。JNDI 名前空間の contexts ディレクトリの下の planet というディレクトリが、この planet というコンテキストに相当します。

新規に Web ドメインを作成した場合、コンテキストのディレクトリ（/webdomains/contexts）に、default というコンテキストが自動的に生成されます。planet と mike というコンテキストを新たに作成した後の contexts ディレクトリがリスト 1-3 のようになります。

```
$ cd contexts
$ ls
default/   planet/   mike/
```

リスト 1-3. contexts ディレクトリ

リクエストの URL で（「/」単位で）最も長く一致するコンテキスト・パスを持

つコンテキストがリクエストに対する処理を行いますが、URL に全く一致する部分がない場合は、この default コンテキストが処理を行います。

1-3-4. コンテキストの設定情報

サーブレット・コンテキストのディレクトリには、コンテキストの設定情報を持つ config オブジェクトや、静的コンテンツのファイルシステム上の位置を持つ doc_root、このコンテキストで公開されているサーブレットのオブジェクトを持つ named_servlets ディレクトリなどがあります（リスト 1-4）。

```
$ cd planet
$ ls
config          httpSecurity
doc_root        named_servlets/
```

リスト 1-4. コンテキストのディレクトリ

config オブジェクトのプロパティがリスト 1-5 のようになります。config オブジェクトの context.servlets グループは、サーブレット・パスと named_servlets ディレクトリ内のサーブレットオブジェクトとのマッピング情報を持っています（リスト 1-5）。

```
$ getproperties config
--group--=context.properties
context.browse.dirs=true
context.welcome.names=index.html:index.htm
context.accept.charset=ISO-8859-1
context.accept.language=en
context.default.languages=*
context.default.charsets=*
--group--=context.params
--group--=context.mime
java=text/plain
html=text/html
htm=text/html
body=text/html
--group--=context.servlets
/errors/internal=internalError
/star=dancer
/roland.dj=jaguar
/final=flontier
--group--=context.error.uris
401=/system/errors/401.html
403=/system/errors/403.html
404=/system/errors/404.html
406=/system/errors/406.html
```

```
500=/errors/internal
```

リスト 1-5. config のプロパティ

doc_root オブジェクトには、FSContextURL というパラメータがあり、このパラメータに静的コンテンツを配置しているファイルシステム上のパスを指定します（リスト 1-6）。

```
$ getproperties doc_root
FSContextURL=/mnt/public_html
```

リスト 1-6. doc_root のプロパティ

これらのプロパティの値を変更するには、「1.2 OSE の設定を行うツール」で解説したコマンドを使用します。例えば、/star というサーブレット・パスに対応するサーブレットを、dancer から jaguar に変更するには、一旦プロパティを削除して、新たにまた追加します（リスト）。

```
$ removegroupentry config context.servlets /star
$ getgroup config context.servlets
/errors/internal=internalError
/roland.dj=jaguar
/final=flontier
$ addgroupentry config context.servlets /star jaguar
$ getgroup config context.servlets
/errors/internal=internalError
/roland.dj=jaguar
/final=flontier
/star=jaguar
```

リスト 1-7. プロパティ・グループ内のプロパティの更新

doc_root の FSContextURL プロパティを変更するには、リスト 1-8 のようにします。

```
$ setproperties doc_root "FSContextURL=/mnt/publich_html/planet"
$ getproperties doc_root
FSContextURL=/mnt/publich_html/planet
```

リスト 1-8. プロパティの更新

1-3-5. サーブレットの設定情報

各コンテキストのディレクトリには、named_servlets というサブディレクトリがあり、このディレクトリに公開されているサーブレットのオブジェクトが格納されています。各サーブレットのオブジェクトは、servlet.class というパラメータを持ち、このパラメータの値が loadjava ユーティリティでロードされたクラスになります。

```
$ cd named_servlets
$ ls
dancer    flontier    jaguar     internalError
$ getproperties dancer
servlet.class=SCOTT:Martian
```

リスト 1-7.　named_servlets ディレクトリ

サーブレットで Init パラメータを使用する方法は、「4.2 Init パラメータ」を参照してください。

1-4. リクエスト処理のしくみ

ここでは、リクエストがあった際に、OSE でどのような処理が行われるかを解説します。

1-4-1. URL の構成

OSE は、リクエストがあると、その URL に適したサーブレットを検索します。その際には、

①Web ドメイン
②サーブレット・コンテキスト
③サーブレット

の順で検索していきます。例えば、次のような URL のリクエストがあったとします。

```
http://cavist.com:7700/red/star?year=1989
```

OSE は、ホスト名（cavist.com）から Web ドメインを検索し、該当する Web ドメインの設定情報から、コンテキスト・パス（/red）に対応するサーブレット・コ

ンテキストを検索します。該当するサーブレット・コンテキストが見つかると、次にサーブレット・パス（/star）に対応するサーブレットをそのコンテキスト内で検索します。また、クエリー・ストリング（year=1989）で指定されているパラメータをサーブレットで使用することができます。

図 1-3. URL の構造

ここで使用される設定情報などは、すべて JNDI 名前空間に格納されていて、セッション・シェル（後述）を使用して管理することができます。

1-4-2. サーブレットの検索手順

では、ここまでの内容をもとに、

```
http://yourhost:7700/red/star
```

というリクエストに対して、どのようにしてサーブレットを検索するかを説明します。

①. yourhost:7700 の部分は、ここではシングル・ドメイン・Web サービスとしているので（リスト 1-1）、このポートに対応する Web サービスである JNDI 名前空間の/service/webserver を使用します。

②. Web ドメインとしてルートディレクトリ/webdomains が該当します。

③. Web ドメインの設定から、最も長く URL が一致するコンテキスト・パスを検索します。/webdomains/config オブジェクトのプロパティ（リスト 1-2）から、/red というコンテキスト・パスを持つ planet コンテキストが該当します。

④. planet コンテキスト内で公開されているサーブレットの中から、/star というサーブレット・パスを持つサーブレットを検索します。/webdomains/contexts/planet/config オブジェクトのプロパティ（リスト 1-5）から、/star というサーブレット・パスは、dancer というサーブレットにマップされていることがわかります。

⑤. /webdomains/contexts/planet/named_servlet にある dancer オブジェクトのプロパティ（リスト 1-7）から、このサーブレットは SCOTT スキーマの Martian クラスにマップされていることがわかり、このクラスがインスタンス化され、リクエストに対する処理を行います。

1-5. OSE の外部構成

OSE は、次の 3 種類の実行形態を取ることができます。

・ディスパッチャに直接接続する構成
・リスナーを経由する構成
・mod_ose を経由する構成

それぞれの構成について解説していきます。

OSE を動作させる場合は、MTS 構成にする必要があります。

1-5-1.ディスパッチャに直接接続する構成

ブラウザはディスパッチャに直接接続します。ディスパッチャは受け取ったリクエストを空いているサーバープロセス(Oracle8*i*JVM) に渡します（図 1-4)。この方法は、構造は単純ですが、１つのディスパッチャがすべてのリクエストを受け付けるため、スケーラビリティに優れません。

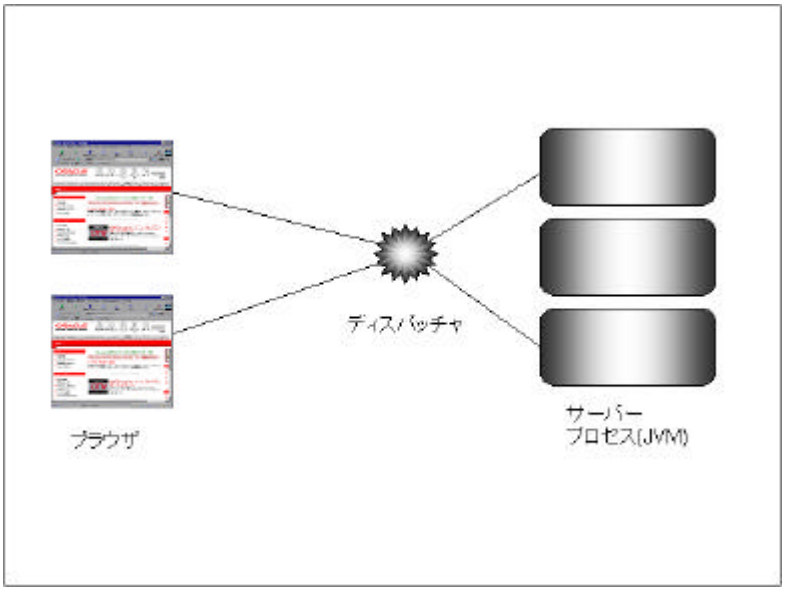


図 1-4. ディスパッチャに直接接続する構成

1-5-2. リスナーを経由する構成

ブラウザは、リスナーに接続します。リスナーは、空いているディスパッチャにクライアントとの接続（ソケット）を渡し、ディスパッチャがサーバープロセスに要求を投げます（図 1-5)。この構成を採ると、ディスパッチャとサーバープロセスの 2 段階で負荷が分散されるため、スケーラビリティを向上させることができます。

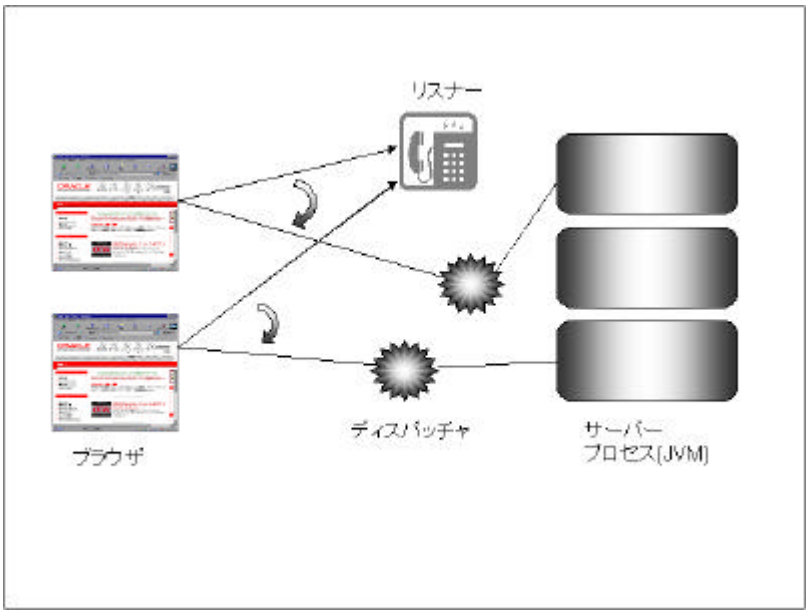


図 1-5. リスナーを経由する構成

1-5-3.mod_ose を経由する構成

mod_ose を使用すると、OSE のフロント・エンドとして Apache(Oracle HTTP Server)を利用できます。mod_ose を使用した構成の場合、ブラウザはまず Apache に接続します。Apache プロセス内部で動作する mod_ose からリスナーに接続し、その後ディスパッチャに接続します（図 1-6）。mod_ose からリスナーに接続する際のプロトコルは通常のデータベース接続と同じプロトコルで、このプロトコルの上に HTTP を被せています。これを **HTTP/Net8**(HTTP over Net8)と呼んでいます。

OSE は、Apache の背後で動作するサーブレットコンテナとして使用するように設計されています。mod_ose を使用することによって、静的なコンテンツを Apache から提供し、動的なコンテンツは OSE から提供することができます。従ってこの構成は、コンテンツの最適な配置が可能になり、もっともスケーラブルな構成となります。

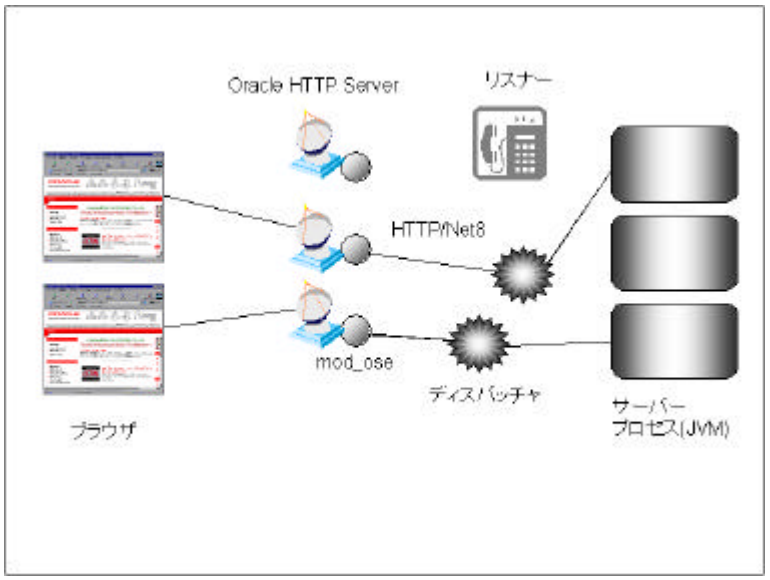


図 1-6. mod_ose を使用する構成

1-5-4. MTS と OSE

Oracle8*i* JVM は、MTS の仕組みを利用することで、スケーラブルなサーバーになりますが、OSE も例外ではありません。OSE は JVM のアーキテクチャの恩恵を十分に受けています。

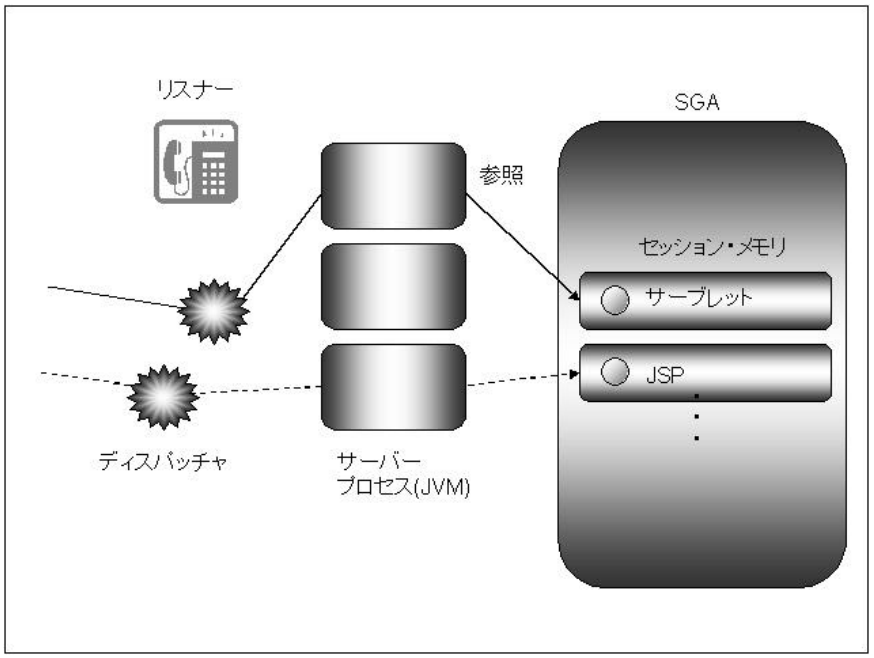


図 1-7. MTS と OSE

図 1-7 のように、リスナーを介して接続されたディスパッチャは、空いているサーバープロセスに要求を投げます（正確には、**要求キュー**と呼ばれるキューに要求を投げ、空いているサーバープロセスがキューから 1 つ要求を取り出し、処理します）。従って、要求がどのサーバープロセスによって処理されるかは、実行時までわかりません。しかし、SGA 内では、各クライアント（ブラウザ）ごとに専用のセッションメモリが確保され、サーバープロセスはそのメモリを使用するため、クライアントにはあたかも自分専用の JavaVM が起動しているように見えます。これを**仮想 JavaVM** と呼びますが、これにより、高いスケーラビリティを達成することができます。

2.環境設定

ここでは、Oracle Servlet Engine の設定と Oracle HTTP Server(Apache)で mod_ose を使用する際の設定方法を解説します。OSE の構成は 3 種類ありますが、ここでは、

・リスナーを経由する構成

・mod_ose を経由する構成

の 2 つについて解説します。ディスパッチャに直接接続する設定は、「5-1 ディスパッチャに直接接続する方法」を参照してください。

OSE の設定は、

①JNDI 名前空間での Web サービスの作成(2-1)

②各パラメータファイルの設定(2-2, 2-3)

の順で行います。①の作業は mod_ose を使用する場合も使用しない場合も共通です。②のパラメータファイルの設定は、リスナー経由で接続する場合は 2-2 を、mod_ose 経由で接続する場合は 2-3 を参照してください。

2.1 Web サービスの作成

はじめに、Web サービスを作成し、JVM 内に Web サーバーの準備をします。

OSE の設定をするには、セッション・シェルを使用します。次のように入力します。

```
% sess_sh –u sys/change_on_install –s jdbc:oracle:oci8:@
```

-s オプションには、JDBC の接続文字列または IIOP の接続文字列を指定します。

Web サービスや Web ドメインの管理は SYS ユーザーで行うことを推奨します。

以降、セッション・シェル上で作業を行ないます。

2-1-1. Web サービスの作成

まず、createwebservice コマンドで、Web サービスを作成します。Web サービスのルートディレクトリおよびサービス名を指定します。

```
$ createwebservice –root <JNDI ロケーション> <Web サービス名>
```

-net8 オプションは、エンドポイントがリスナーからの接続であることを表します。

例えば、次のようにします。

```
$ createwebservice –root /webdomains webserver
```

これにより、/service/webserver という Web サービスオブジェクトおよび、/webdomains ディレクトリが作成・初期化されます。また、この Web サービスのルートディレクトリは/webdomains になります。

マルチ・ドメインサービスを作成する場合は、複数の IP アドレスを使用する際は-ip オプション、仮想ホストを使用する際は、-virtual オプションを指定してください。詳しくは、「5-3. マルチ・ドメイン Web サービスの設定」を参照してください。

2-1-2. エンドポイントの作成

次に、addendpoint コマンドを使用して、エンドポイントを登録します。

```
$ addendpoint –net8 [–timeout <タイムアウト>] <Web サービス名> <エンドポイント名>
```

例えば、次のように入力します。

```
$ addendpoint –net8 –timeout 300 webserver endpnet8
```

2-1-3. Web ドメインの作成

次に、createwebdomain コマンドを使用して、新しい Web ドメインを作成します。

```
$ createwebdomain [–docroot <ファイルディレクトリ>] <Web ドメイン名(JNDI)>
```

「-docroot」オプションでは、デフォルト・コンテキストの静的コンテンツの場所を指定することができます。例えば、次のように入力します。

```
$ createwebdomain –docroot  /mnt/public_html /webdomains
```

また、エラーページなどを表示できるように、ドキュメント・ルートに、エラーページのファイルをコピーします。

```
% cp –r $ORACLE_HOME/jis/admin/public_html/system /mnt/public_html
```

また、OSE の動作をあとで確認するために、/mnt/public_html に適当な index.html を置いておきます。

そして、セッション・シェルを終了します。

```
$ exit
```

2-2. リスナー経由で接続する場合のパラメータファイルの設定

2-2-1. 初期化パラメータファイルの編集

初期化パラメータファイル（init.ora）に MTS_DISPATCHERS パラメータを追加します。

```
MTS_DISPATCHERS = “(PROTOCOL=TCP)(DISP=5)”
```

DISP には、起動するディスパッチャの数を指定します。デフォルトは 1 です。

通常のデータベース接続用にデフォルトのポート 1521 を使用しない TCP/IP リスナー・アドレスを構成する場合には、初期化パラメータファイルに次のように local_listner パラメータを設定します。

```
local_listner="(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
                                      (HOST=localhost)
                                      (PORT=<ポート番号>)))"
```

2-2-2. リスナーの設定

LISTENER.ORA ファイルにエントリを追加します。

```
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = <ポート番号>))
  (PROTOCOL_STACK =
    (PRESENTATION = http://<Web サービス名>)
  )
)
```

<ポート番号>は、HTTP プロトコルをリスニングするポート番号になります。例えば、次のようにします。

```
ORA817 =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = anago)(PORT = 1521))
    )
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC))
    )
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = anago)(PORT = 2481))
      (PROTOCOL_STACK =
        (PRESENTATION = GIOP)
        (SESSION = RAW)
      )
    )
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = anago)(PORT = 7700))
      (PROTOCOL_STACK =
        (PRESENTATION = http://webserver)
      )
    )
  )
```

リスナーを再起動し、データベースを再起動します。
ブラウザから、

```
http://yourhost:7700/index.html
```

にアクセスして確認します。

データベースに接続しているセッションがタイムアウトせずに残っていると、データベースをシャットダウンできないことがあります。この場合は次のように immediate 引数をつけてシャットダウンします。

```
SVRMGR> shutdown immediate
```

2-3. mod_ose を経由して接続する場合の設定

mod_ose の設定を行なう前に、OSE 内に Web アプリケーションを作成・公開する必要があります。これは、mod_ose のための設定ファイルが、OSE 内部の設定から自動生成されるためです。この項では、「3 チュートリアル」を実行した後、mod_ose を設定するものとして解説を行ないます。

2-3-1. 初期化パラメータファイルの編集

初期化パラメータファイル（init.ora）に MTS_DISPATCHERS パラメータを追加します。

```
MTS_DISPATCHERS = “(PROTOCOL=TCP)(DISP=5)”
```

DISP には、起動するディスパッチャの数を指定します。デフォルトは 1 です。初期化パラメータファイルの編集後、データベースを再起動します。

データベースに接続しているセッションがタイムアウトせずに残っていると、データベースをシャットダウンできないことがあります。この場合は

次のように immediate 引数をつけてシャットダウンします。

```
SVRMGR> shutdown immediate
```

通常のデータベース接続用にデフォルトのポート 1521 を使用しない TCP/IP リスナー・アドレスを構成する場合には、初期化パラメータファイルに次のように local_listner パラメータを設定します。

```
local_listner="(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
                                       (HOST=localhost)
                                       (PORT=<ポート番号>)))"
```

2-3-2. リスナーの設定

リスナーの設定は必要ありません。通常のデータベース接続の設定がされていれば十分です。

2-3-2. tnsnames.ora の設定

mod_ose は、network/admin/tnsnames.ora からリスナーの位置を取得し、リスナーに接続しようとします。このときの様子を表したものが、図 2-2 になります。

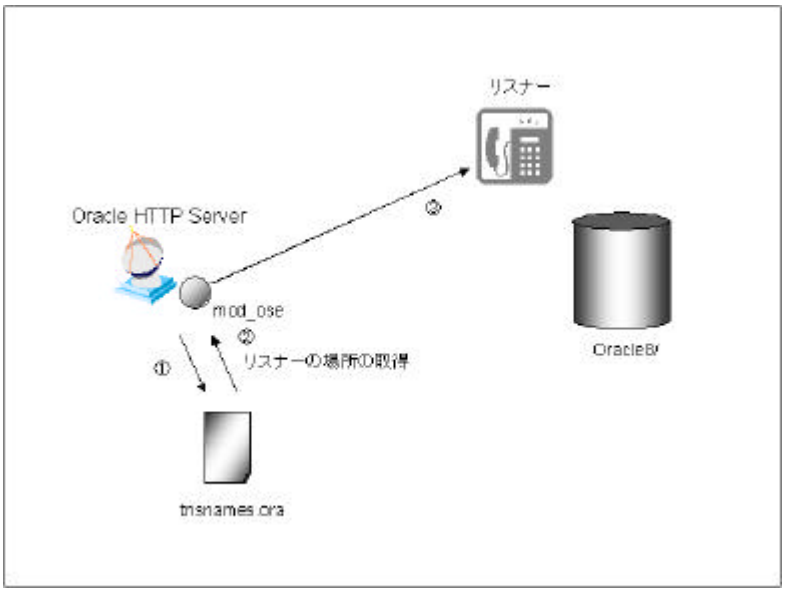


図 2-2. mod_ose の構成

tnasnames.ora に Oracle HTTP Server からリスナに接続するための接続情報を記

述したエントリを追加します。書式は以下のようになります。

```
<エントリ名> =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = ホスト名)(PORT = リスナーポート番
号))
    (CONNECT_DATA =
      (SERVICE_NAME = <SERVICE_NAME>)
      (SERVER = SHARED)
      (PRESENTATION = http://<Web サービス名>)
    )
  )
```

例えば、次のようにします。

```
MODOSE_HTTP =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA =
      (SERVICE_NAME = ora817)
      (SERVER = SHARED)
      (PRESENTATION = http://webserver)
    )
  )
```

2-3-4. mod__ose.conf の設定

次に、セッション・シェルで、Web ドメインの設定情報を取り出します。設定情報の取り出しは、exportwebdomain コマンドで行います。

```
$ exportwebdomain –format apache <Web ドメイン(JNDI)> &> <ファイル名>
```

“&>”は、セッションシェルのコマンドの出力を、オペレーティングシステムのファイルにリダイレクトするためのものです。

例えば、次のようにします。

```
$ exportwebdomain –format apache /webdomains &> /tmp/mod__ose_tmp.conf
```

ここで生成されたファイル（mod__ose_tmp.conf）を基にして、$ORACLE_HOME/Apache/Apache/conf にある mod_ose の設定ファイル

（mod__ose.conf）を編集します。リスト 2-1 は、編集前のもので、これに変更を加えます。

```
 1   LoadModule ose_module      libexec/libjipa8i.so
 2
 3   #
 4   # Apache configuration
 5   # Domain: /webdomains
 6   #
 7   # o/p generated by
 8   # exportwebdomain -format apache -netservice inst1_http -nodocs
-nodefault /webdomains
 9   #
10
11   <IfModule mod_ose.c>
12
13   AuroraService inst1_http
14
15   #
16   # Context for VPATH /context-test/
17   #
18
19   <Location /context-test/ >
20   AddHandler aurora-server snoop
21   </Location>
22
23   <Location /context-test/admin/shell >
24   SetHandler aurora-server
25   </Location>
26
27   <Location /context-test/errors/internal >
28   SetHandler aurora-server
29   </Location>
30
31   <Location /context-test/examples/counter >
32   SetHandler aurora-server
33   </Location>
34
35   <Location /context-test/examples/snoop >
36   SetHandler aurora-server
37   </Location>
38
39   <Location /context-test/servlet/* >
40   SetHandler aurora-server
41   </Location>
42
43   </IfModule>
```

リスト 2-1. mod__ose.conf

13 **行目：**

次のように、tnsnames.ora **に追加したエントリ名に変更します。**

```
AuroraService <TNSNAMES エントリ名>
```

例えば、次のようにします。

```
AuroraService MODOSE_HTTP
```

14 **行目：**

ここから 43 **行目の**</IfModule>**までの間に、**exportdomain **コマンドで作成した設定ファイル（**mod__ose_tmp.conf**）の中の必要な部分だけ移します。例えば、このファイルには、リスト** 2-2 **のような部分がありますので、この部分を** mod__ose.conf **の** 14 **行目から** 43 **行目までの間にコピーします。この部分は、サーブレットまたは、**JSP **のサーブレット・パスを指定しています。**

```
… 省略 …
<Location /hello >
SetHandler aurora-server
</Location>

<Location /hellouser.jsp >
SetHandler aurora-server
</Location>

<Location /errors/internal >
SetHandler aurora-server
</Location>
… 省略 …
```

リスト 2-2. mod__ose_tmp.conf

この後、Oracle HTTP Server **を再起動して、**

```
http://yourhost:7777/hello
```

```
http://yourhost:7777/hellouser.jsp
```

にアクセスして、確認します。リスナーのポート番号が 7700 であるのに対し、Oracle HTTP Server のポート番号は 7777（デフォルト）であるので、Oracle HTTP Server に来たリクエストが mod_ose を経由して、OSE に渡されているのが確認できます。

ここでの exportwebdomain コマンドを使用する設定は、「2.1 Web サービスの作成」を終了していることが前提になります。

Oracle HTTP Server は、JSP のリクエストの処理に、デフォルトで Apache JServ 上の Oracle JSP を使用するように設定されています。OSE にリクエストを渡すためには、$ORACLE_HOME/Apache/jsp/conf にある ojsp.conf 内の 2 行をコメントアウトしてください。

```
#ApJServAction .jsp /servlets/oracle.jsp.JspServlet
#ApJServAction .sqljsp /servlets/oracle.jsp.JspServlet
```

リスト 2-3. ojsp.conf

先ほど、サーブレット・パスの設定を mod__ose.conf に追加しましたが、この部分にはワイルドカードも使用することができるので、次のようにすることで、/ose 以下のリクエスト全てを OSE に渡すことができます。これは、あるコンテキストごとに OSE を使用する、しないの設定をするのに便利です。

```
<Location /ose/* >
SetHandler aurora-server
</Location>
```

3.チュートリアル

ここでは、OSE のみでサーブレットと JSP を使用する方法を解説します。mod_ose を使用する場合、URL のポート番号がリスナーのポート番号（この章では 7700）から、Oracle

HTTP Server のポート番号（デフォルトで 7777）に変更するだけで、サーブレットと JSP の公開の方法は同じになります。例えば、

リスナー経由で接続する場合（2-1 の設定）

http://yourhost:7700

ディスパッチャに直接接続する場合（5-1 の設定）

http://yourhost:8800

mod_ose 経由で接続する場合（2-2 の設定）

http://yourhost:7777

になります。

mod_ose を使用する場合は、Oracle HTTP Server 側でもコンテキスト・パスの設定をする必要がありますので、チュートリアルの作業の実行後に「2.2 mod_ose の設定」を参照して mod_ose の設定を行なってください。

3.1 サーブレットの実行

サーブレットを実行するために、以下の手順を行います。

①サーブレットのコンパイル(javac コマンド)

②クラスのデータベースへのロード(loadjava コマンド)

③サーブレットの公開(publishservlet コマンド)

まず、サーブレットをコンパイルします。ここでは、Oracle8*i* 付属のサンプルに含まれるサーブレットである HelloServlet.java を使用します。

```
% cd $ORACLE_HOME/javavm/demo/examples/servlets/basic/helloworld
% javac HelloServlet.java
```

サーブレットのコンパイルに必要な環境設定については、10 章の「2. 環境設定」を参照してください。

次に、loadjava コマンドを使用して、クラス（HelloServlet.class）をデータベースにロードします。

```
% loadjava –u scott/tiger –r –v –f HelloServlet.class
initialization complete
loading   : HelloServlet
```

```
creating  : HelloServlet
resolver  :
resolving : HelloServlet
```

セッション・シェル上で、サーブレットを公開します。公開には、publishservlet コマンドを使用します。

```
$ publishservlet –virtualpath <サーブレット・パス> <コンテキスト名(JNDI)>
<サーブレット名(JNDI)> <クラス>
```

例えば、次のようにします。

```
% sess_sh –u sys/change_on_install –s jdbc:oracle:oci8:@
--Session Shell–
--type ”help” at the command line for help message
$ publishservlet –virtualpath /hello /webdomains/contexts/default ¥
> helloServlet SCOTT:HelloServlet
$ exit
```

Net8 リスナー経由の場合、ブラウザから、

```
http://yourhost:7700/hello
```

と入力すると、図 3-1 のように出力されます。

図 3-1. 出力結果

3.2 JSP の実行

OSE で JSP を実行する方法を解説します。OSE で JSP を実行するために、以下の手順を行います。

①JSP ファイルのデータベースへのロード(loadjava コマンド)

②JSP の公開(publishjsp コマンド)

まず、loadjava コマンドを使用して、JSP ファイルをデータベースにロードします。ここでは、Oracle8*i* に付属のサンプルに含まれる hellouser.jsp を使用します。

```
% cd $ORACLE_HOME/jsp/demo/demo/basic/hellouser
% loadjava –u scott/tiger –v hellouser.jsp
initialization complete
loading   : hellouser.jsp
creating  : hellouser.jsp
```

セッション・シェル上で、JSP を公開します。公開には、publishjsp コマンドを使用します。

```
publishjsp –schema <スキーマ名> –context <コンテキスト名(JNDI)> <JSP ファイル名>
```

例えば、次のようにします。

```
% sess_sh –u sys/change_on_install –s jdbc:oracle:oci8:@
--Session Shell–
--type ”help” at the command line for help message
$ publishjsp –schema SCOTT –context /webdomains/contexts/default ¥
> hellouser.jsp
$ exit
```

-context オプションは、コンテキストを指定するオプションです。このオプションを省略すると default コンテキストに設定されますので、上の例では、このオプションを省略しても同じ結果になります。

Net8 リスナー経由の場合、ブラウザから、

```
http://yourhost:7700/hellouser.jsp
```

と入力すると、図 3-2 のように出力されます。

図 3-2. 出力結果

3-3. データベースへのアクセス

データベースにアクセスする Web アプリケーションの場合、データベースにアクセスするユーザーは Web ドメインのオーナーになります。したがって、これまでの設定のままでは、SYS ユーザーでデータベースにアクセスすることになります。データベースにアクセスする場合は、Web ドメインのオーナーを適切なユーザーに変更する必要があります。例えば、次のようにして SCOTT に変更します。

```
% sess_sh -s jdbc:oracle:oci8:@ -u sys/change_on_install
--Session Shell--
--type "help" at the command line for help message
$ ls -l
Read      Write     Exec      Owner     Date Time      Type
Name
PUBLIC    SYS       PUBLIC    SYS       Sep 26 23:09   Context
bin
PUBLIC    SYS       PUBLIC    SYS       Sep 26 23:09   Context
etc
SYS       SYS       SYS       SYS       Sep 26 23:09   Context
service
SYS       SYS       SYS       SYS       Sep 26 23:10   Context
system
PUBLIC    PUBLIC    PUBLIC    SYS       Sep 26 23:09   Context
test
SYS       SYS       SYS       SYS       Sep 29 13:19   Context
webdomains
$ chown -R SCOTT webdomains
$ ls -l
```

```
Read      Write     Exec      Owner     Date Time      Type
Name
PUBLIC    SYS       PUBLIC    SYS       Sep 26 23:09   Context
bin
PUBLIC    SYS       PUBLIC    SYS       Sep 26 23:09   Context
etc
SYS       SYS       SYS       SYS       Sep 26 23:09   Context
service
SYS       SYS       SYS       SYS       Sep 26 23:10   Context
system
PUBLIC    PUBLIC    PUBLIC    SYS       Sep 26 23:09   Context
test
SYS       SYS       SYS       SCOTT     Sep 29 13:19   Context
webdomains
$ exit
```

Web ドメインのオーナーを変更したら、データベースにアクセスするプログラムを書きます。ここでは、JSP ファイルからデータベースにアクセスしてみます。

```
<%@page contentType="text/html;charset=Shift_JIS"%>
<%@page import="java.sql.*"%>
<HTML>
 <BODY>
  Employees:
  <TABLE BORDER=1 CELLPADDING=5>
    <TR>
      <TH>従業員番号</TH>
      <TH>名前</TH>
    </TR>
    <%
      Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection conn =
          DriverManager.getConnection("jdbc:oracle:kprb:", "demo",
"demo");
      Statement stmt = conn.createStatement();
     ResultSet rset = stmt.executeQuery("select empno, ename from emp");
      while (rset.next()) {
    %>
        <TR>
         <TD><%= rset.getString("empno") %></TD>
         <TD><%= rset.getString("ename") %></TD>
        </TR>
    <%
      }
      rset.close();
      stmt.close();
      conn.close();
    %>
  </TABLE>
 </BODY>
</HTML>
```

リスト 3-1　db.jsp

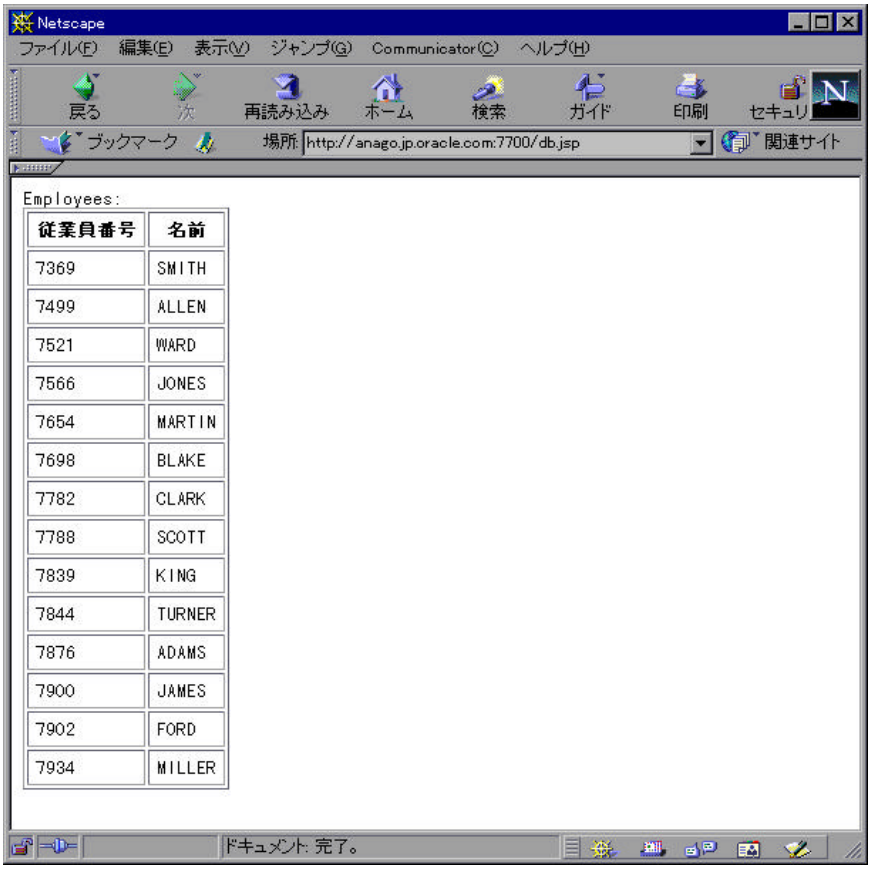
JDBC サーバー側内部ドライバを使用するため、接続文字列が jdbc:oracle:kprb となっていることに注意してください。この JSP ファイルをデータベースにロードします。

```
loadjava –v –u scott/tiger db.jsp
```

次に、セッション・シェルを用いて SCOTT ユーザーでログインし、JSP を公開します。

```
% sess_sh -s jdbc:oracle:oci8:@ -u scott/tiger
--Session Shell--
--type "help" at the command line for help message
$ publishjsp -context /webdomains/contexts/default db.jsp
$
```

ブラウザからアクセスすると、図 3-3 のように表示されます。



従業員番号	名前
7369	SMITH
7499	ALLEN
7521	WARD
7566	JONES
7654	MARTIN
7698	BLAKE
7782	CLARK
7788	SCOTT
7839	KING
7844	TURNER
7876	ADAMS
7900	JAMES
7902	FORD
7934	MILLER

図 3-3. db.jsp の実行結果

4.高度なトピック

4-1. OSE 管理用コマンド

ここでは、OSE で JNDI 名前空間を管理するためのコマンドを解説します。各コマンドの詳細は、「Oracle8*i* Java Tools リファレンス」マニュアルを参照してください。

4-1-1. Web サービス管理コマンド

createwebservice

新しく Web サービスを作成します。

```
$ createwebservice –root <JNDI ロケーション> <Web サービス名>
```

destroyservice

サービスを削除します。次のように使用します。

```
$ destroyservice <サービス名>
```

addendpoint

Web サービスのエンドポイントを追加します。

```
$ addendpoint –net8(-port) [–timeout <タイムアウト>] <Web サービス名> <
エンドポイント名>
```

-timeout オプションで、ソケットのタイムアウトを指定します。

rmendpoint

エンドポイントを削除します。

```
$ rmendpoint <Web サービス名> <エンドポイント名>
```

4-1-2. Web ドメイン管理コマンド

createwebdomain

新しく Web ドメインを作成します。

```
$ createwebdomain [–docroot <ファイルディレクトリ>] <Web ドメイン名(JNDI)>
```

-docroot オプションは、この Web ドメインのデフォルト・コンテキストの静的コンテンツのある（ファイル・システム上の）ディレクトリを指定します。

destroywebdomain

Web ドメインを削除します。

```
$ destroywebdomain <Web ドメイン名(JNDI)>
```

4-1-3. サーブレット・コンテキスト管理コマンド

createcontext

新しくサーブレット・コンテキストを作成します。createcontext コマンドは、次のように使用します。

```
$ createcontext -virtualpath <コンテキスト・パス> -docroot <ファイルディレクトリ> <Web ドメイン名(JNDI)> <コンテキスト名>
```

-virturalpath オプションで、コンテキスト・パスを指定します。-docroot オプションで、このコンテキストの静的コンテンツのディレクトリを指定します。例えば、次のように指定します。

```
$ createcontext -virtualpath /red -docroot /mnt/public_html /webdomains planet
```

上の例では、webdomains という Web ドメイン上で、planet というコンテキストを作成しています。この際、静的コンテンツは全て /mnt/public_html にあり、サーブレットにアクセスする際に使用するサーブレット・パスは/red になります。

destroycontext

サーブレット・コンテキストを削除します。登録されているサーブレットも削除されます。

```
$ destroycontext <コンテキスト名(JNDI)>
```

4-1-4. サーブレット管理コマンド

publishservlet

サーブレットを、指定したサーブレット・コンテキストで公開します。

```
$ publishservlet -virtualpath <サーブレット・パス> <コンテキスト名(JNDI)>
<サーブレット名> <クラス>
```

例えば、次のようにします。

```
$ publishservlet -virtualpath /star /webdomains/contexts/planet dancer
SCOTT:Martian
```

上の例では、planet コンテキストで SCOTT スキーマの Martian というクラスを dancer というサーブレット名で JNDI 名前空間に公開しています。サーブレット・パスは/star で、planet コンテキストのコンテキスト・パスは/red ですので、URL からこのサーブレットを呼び出す場合は、次のようになります。

```
http://yourhost:7700/red/star
```

unpublishservlet

コンテキストからサーブレットを削除します。

```
$ unpublishservlet <コンテキスト名(JNDI)> <サーブレット名>
```

4-1-5. JSP 管理コマンド

publishjsp

JSP からサーブレットへの変換、コンパイル、公開を一括して行うことができます。JSP ファイル名には、loadjava コマンドによってロードされた JSP のファイル名を指定します。

```
$ publishjsp –schema <スキーマ名> -context <コンテキスト名(JNDI)> <JSP ファイル名>
```

unpublishjsp

JSP ファイルを JNDI 名前空間から削除します。

```
$ unpublishjsp –context <コンテキスト名(JNDI)> <JSP ファイル名>
```

4-1-6. Export 用コマンド

exportwebdomain

mod_ose などのための設定ファイルを生成することができます。

```
$ exportwebdomain –format <フォーマット> <Web ドメイン名> &> <ファイル名>
```

-format オプションでは、mod_ose 用の場合は apache、IIS 用の場合は iis と指定することで、それぞれに対応した設定ファイルを生成することができます。

4-2. サーブレットの init パラメータ

サーブレットを使用するには、まず loadjava コマンドを使用してサーブレットのクラスをデータベースにロードし、次にセッション・シェルで publishservlet コマンドを使用して公開します。もし、公開したサーブレットで init パラメータを使用する場合は、setproperties コマンドを使用して init パラメータを設定します。以下に、その手順を解説します。

まず、loadjava コマンドでサーブレットをロードします。

```
% loadjava –u scott/tiger –r –v –f Martian.class
initialization complete
loading   : Martian
creating  : Martian
resolver  :
resolving : Martian
```

これで、Martian というクラスが SCOTT スキーマにロードされました。

次に、このサーブレットをセッション・シェルを使用して公開します。

```
% sess_ss –u sys/change_on_install –s jdbc:oracle:oci8:@
--Session Shell—
```

```
--type "help" at the command line for help message
$ publishservlet -virtualpath /star /webdomains/contexts/planet ¥
> dancer SCOTT:Martian
$ exit
```

ここでは、SCOTT スキーマの Martian というクラスを/webdomains/contexts/planet コンテキストで、dancer というサーブレットとして公開しています。このサーブレットのサーブレット・パスは/star になっています。

このサーブレットで init パラメータを使用する場合は、setproperties コマンドを使用して設定します。

```
$ cd webdomains/contexts/planet/named_servlets
$ getproperties dancer
servlet.class=SCOTT:Martian
$ setproperties dancer "servlet.class=SCOTT:Martian
> area=313
> fire=keeper"
$ getproperties dancer
servlet.class=SCOTT:Martian
area=313
fire=keeper
```

init パラメータを設定することによって、HttpServlet クラスの getInitParameter() メソッドで、設定したパラメータと値を取得することができます。

4-3.ステートフルな Web アプリケーションとステートレスな Web アプリケーション

Web アプリケーションの実装には、ステートフルなものとステートレスなものがあります。ステートフルな Web アプリケーションでは、サーバー側（正確にはサーバー側 JVM）に状態を残し、次のリクエストでその状態を引き継ぐことができます。例えばサーバー側の JVM 内にカウンタを用意し、同じクライアントからのアクセス数をカウントするようなアプリケーションはステートフルと言えます。また、ショッピングカートをサーバー側の JVM 内に持つアプリケーションでは、ページを移動しても買い物かごの中身を保つため、**ステートフル**な Web アプリケーションと言えます。

一方、ステートレスなアプリケーションはサーバー側に状態を残しません。例えば、検索エンジンのようにユーザーの入力に対し、１回きりの処理を行なうようなアプリケーションはステートレスです。また、カウンタの値やショッピングカートの内容をデータベースに保存し、リクエストのたびにデータベースから状

態を復元するような実装ならば、サーバー側の JVM に状態を残さないため**ステートレス**なアプリケーションと言えます。

ここでのステートフル／ステートレスの議論は、Oracle JVM の立場での議論です。すなわち、ユーザーから見て（論理的に）ステートフルかステートレスかということではないので、注意してください。（ユーザーから見てステートフルでも、プログラムの作りがステートレスなことがあります）

サーブレットや JSP プログラミングにおいては、(Oracle JVM の立場での)ステートフルかステートレスかの違いは、そのサーブレットや JSP が HTTPSession を使用するかどうかで決定されます。

ステートレスな設定にするためには、サーブレットや JSP を公開する際に -stateless オプションを指定します。例えば

```
$ publishservlet –statless –virtualpath ….
$ publishjsp –statless –schema …
```

のようにします。また、このようにステートレスに設定された状態で mod__ose.conf 用の設定を exportwebdomain コマンドで出力すると、以下のように SetHandler が「aurora-stateless-server」になって出力されます。

```
<Location /hello3.jsp >
SetHandler aurora-stateless-server
</Location>
```

一般的に、ステートレスなサーブレットの方がステートフルなサーブレットよりもメモリの使用量が少なくてすみます。ステートレスなサーブレットは、あるクライアント（ブラウザ）の処理をした後、別のクライアントの処理を行なうことができるため、仮想 JavaVM の使いまわしが可能です。一方、ステートフルなサーブレットは、各クライアントの状態を保つため、クライアント（ブラウザ）ごとに仮想 Java VM を立ちあげなければなりません(図 4-1)。

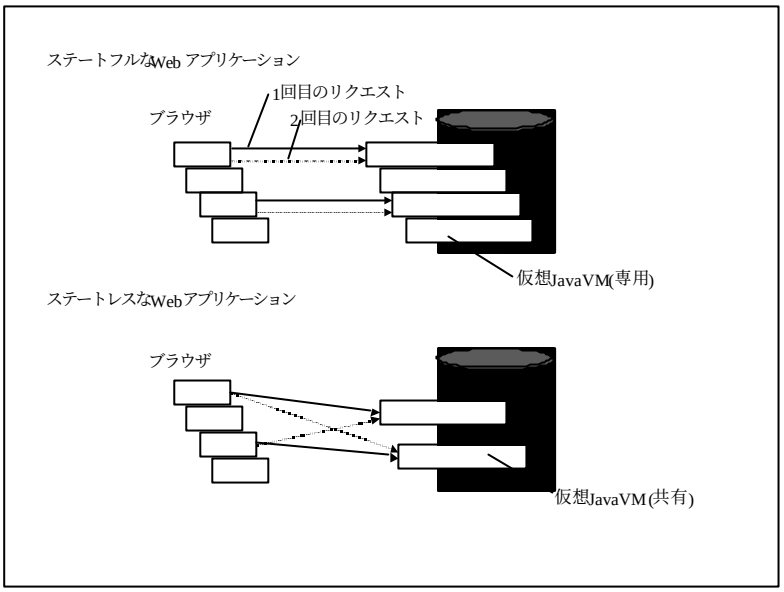


図 4-1. ステートレスとステートフル

従って、ステートレスなアプリケーションはステートレスであることを宣言しておいた方が、Oracle8*i* **のメモリの使用量を最小限に押さえることができます。ただし、**HTTPSession **を使用することはできなくなります。**

5.Tips と注意事項

5-1. ディスパッチャに直接接続する方法

ここでは、ディスパッチャを起動し、クライアントからディスパッチャに直接接続するための設定方法を解説します。

5-1-1. エンドポイントの作成

Web **サービスを作成した後に、ポート番号を指定してエンドポイントを作成します。その場合には、次のようにします。この例では、ポート番号** 8800 **のディスパッチャでアクセス可能になります。**

```
$ addendpoint –port 8800 –timeout 300 webserver endp8800
```

5-1-2. 初期化パラメータファイルの編集

次に、初期化パラメータファイル（init.ora**）を編集します。初期化パラメータファイルに** MTS_DISPATCHERS **パラメータを追加します。**

```
MTS_DISPATCHERS =
   “(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=<ポート番号>))
    (PRE=http://<Web サービス名>)”
```

例えば、次のようにします。

```
MTS_DISPATCHERS =
   “(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=8800))
    (PRE=http://webserver)”
```

データベースを再起動します。

ブラウザから、

```
http://yourhost:8800/index.html
```

にアクセスして確認します。

MTS_DISPATCHERS = “(PROTOCOL=TCP)”の行は必要ありません。また、リスナーの設定も必要ありません。

5-2. admin の設定

OSE には、初期状態で admin という Web サービスが用意されています。この admin という Web サービスを設定することによって、HTTP 経由でセッション・シェルを使用して OSE を管理することができたり、エラー・ログやイベント・ログやドキュメントをブラウザから確認することができます。

まず、ポート番号 8080 で、直接ディスパッチャに接続できるように初期化パラメータファイル（init.ora）を編集します。

```
MTS_DISPATCHERS =
   “(ADDRESS=(PROTOCOL=TCP)(HOST=<ホスト名>)(PORT=8080))
    (DISP=1)(PRE=http://admin)”
```

また、セッション・シェルで接続して、エンドポイントを追加します。

```
$ addendpoint –port 8080 –timeout 300 admin endpoint_admin
```

データベースを再起動して、ブラウザから

```
http://yourhost:8080/
```

にアクセスします。

図 5-2. ブラウザで確認

このページから各ログを確認することができます。

また、HTTP 経由でセッション・シェルを使用することができるようになります。次のようにします。

```
% sess_sh -u sys/change_on_install -s http://yourhost:8080
```

接続後の使用方法は、JDBC 経由で接続したときと同じになります。

5-3. OSE のログ

OSE のログの設定は、デモ・プログラム

$ORACLE_HOME/javavm/demo/examples/web/service

を参考にして行なうことができます。OSE のログには次の 3 種類あります。

イベント･ログ	サーブレットが出力するログ
エラー･ログ	例外など、エラーが発生したときのログ
アクセス･ログ	アクセスした URL のログ

これらのログの設定は以下の順序で行ないます。

①ログを保存する表および順序を作成します

②各ログと、ログを保存する表を関連付けます

③ログ出力用サーブレットを公開し、クライアントからアクセス可能にします

このログの設定は Web サービスごとに行ないます。

OSE 以外の部分でのエラー（ネットワークやセキュリティなどのエラー）は、ここで設定するログには出力されません。これらのエラーを調べるには、初期化パラメータファイル(init<SID>.ora)の background_dump_dest パラメータで指定したディレクトリ上のファイル（ファイル名は {SID}_sXXX_{プロセス番号}.trc）や、$ORACLE_HOME/network/log に出力されるリスナーのログなどが参考になります。

また、mod_ose のログは、Apache のエラーログ（デフォルトでは $ORACLE_HOME/Apache/Apache/logs/httpd_error_log）に出力されます。

5-3-1. ログを保存する表および順序の作成

SCOTT ユーザーなどでログインし、アクセス･ログを保存する表を、以下のように作成します。

```
SQL> CREATE TABLE http$log$ (
       server_name VARCHAR2(80),
       timestamp DATE,
       remote_host RAW(4),
       remote_user VARCHAR2(80),
       request_line VARCHAR2(256),
       status NUMBER(3),
       response_size INTEGER,
       request_method RAW(1),
       referer VARCHAR2(80),
       agent VARCHAR2(80));
```

イベント・ログを保存するための表および順序を、以下のように作成します。

```
SQL> CREATE TABLE event$log ( id NUMBER, line NUMBER, text VARCHAR2(4000) );
SQL> CREATE SEQUENCE event$log_id;
```

同様に、エラーログを保存するための表および順序を、以下のように作成します。

```
SQL> CREATE TABLE error$log ( id NUMBER, line NUMBER, text VARCHAR2(4000) );
SQL> CREATE SEQUENCE error$log_id;
```

5-3-2. 各ログと、ログを保存する表の関連付け

表の関連付けはセッション・シェルを用いて行ないます。

ここでは、Web ドメインが/webdomains JNDI ディレクトリに存在し、この Web ドメインには 2 つのコンテキスト(context1, default)が存在するとします。このとき、アクセス・ログの表の関連付けは以下のように行ないます。

```
$ accesslog -table SCOTT.HTTP$LOG$ /webdomains/context/context1
$ accesslog -table SCOTT.HTTP$LOG$ /webdomains/context/default
```

イベント・ログのための表の対応付けは以下のように行ないます。

```
$ bind /webdomains/servicelogs/event -rebind ¥
   -c SYS:oracle.aurora.namespace.rdbms.TableStream ¥
   -f oracle.aurora.namespace.PublishedObjectFactory ¥
   -string table.name SCOTT.EVENT$LOG
```

エラー・ログのための表の対応付けは以下のように行ないます。

```
$ bind /webdomains/servicelogs/error -rebind ¥
   -c SYS:oracle.aurora.namespace.rdbms.TableStream ¥
   -f oracle.aurora.namespace.PublishedObjectFactory ¥
   -string table.name SCOTT.ERROR$LOG
```

5-3-3. ログ出力用サーブレットの公開

最後に JVM に用意されているログ出力用サーブレットを適当なコンテキスト内でセッション・シェルを用いて公開します。アクセス・ログ用の

サーブレットの公開は、以下のように行ないます。

```
$ publishservlet -virtualpath /http_log context1 httpLog_viewer ¥
    SYS:oracle.aurora.mts.http.servlet.HttpRdbmsLogServlet ¥
    -properties table.name=SCOTT.HTTP$LOG$
```

同様に、イベント・ログ用のサーブレットの公開は以下のように行ないます。

```
$ publishservlet -virtualpath /event_log context1 event_log_viewer ¥
    SYS:oracle.aurora.mts.http.servlet.TableReaderServlet ¥
    -properties table.name=SCOTT.EVENT$LOG
```

エラー･ログ用のサーブレットの公開は以下のように行ないます。

```
$ publishservlet -virtualpath /error_log context1 error_log_viewer ¥
     SYS:oracle.aurora.mts.http.servlet.TableReaderServlet ¥
    -properties table.name=SCOTT.ERROR$LOG
```

5-3-4. クライアントからアクセス

最後に、クライアントからアクセスして動作確認を行ないます。ここでは、ログ用サーブレットを context1 **内に公開したので、このコンテキストのコンテキスト･パスを**/url1 **とすると、アクセス･ログ、イベント･ログおよびエラー･ログの** URL **は**

http://**ホスト名**:**ポート番号**/url1/http_log
http://**ホスト名**:**ポート番号**/url1/event_log
http://**ホスト名**:**ポート番号**/url1/error_log

となります。

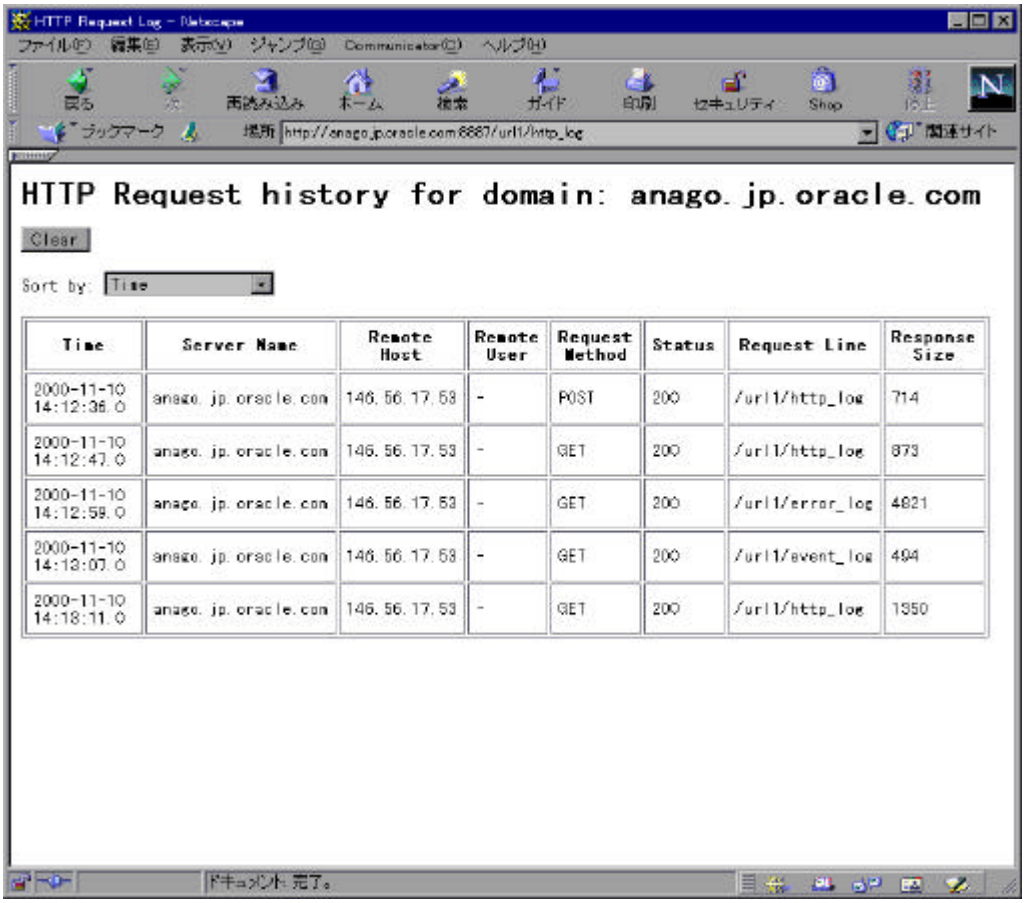


図 5-3.アクセス・ログ

5-4. メモリ不足時の対処

ユーザー数や、オブジェクトのサイズなどによっては、メモリ不足になることがあります。この場合、初期化パラメータファイル(init<SID>.ora)を編集し、JAVA_POOL_SIZE を増やします。

```
JAVA_POOL_SIZE=50000000
```

この例では、JAVA プールのサイズを 50MB に指定しています。(デフォルトは 20MB)

5-5. マルチ・ドメイン Web サービスの設定

マルチ・ドメイン Web サービスでは、JNDI 名前空間上での設定が多少異なります。ここでは、マルチ・Web ドメインサービスでの設定方法とそのときの JNDI 名前空間について解説します。

マルチ・ドメイン Web サービスを作成する場合は、複数の IP アドレスを使用す

る際は-ip **オプション、仮想ホストを使用する際は、**-virtual **オプションを指定する必要があります。例えば、セッション・シェルで次のようにします。**

```
$ createwebservice -root /webdomains -ip -virtual webserver
```

この場合では、IP **アドレスを複数使用し、さらに仮想ホストも使用します。次に、エンドポイントを作成します。**

```
$ addendpoint –net8 –timeout 300 webserver endpnet8
```

次に、1 **つの** IP **アドレス**(10.1.1.1)**に、**1 **つ目の** Web **ドメインを作成します。この際、**Web **サービスのルートディレクトリに、**IP **アドレスの値と同じ名前のディレクトリを作成し、さらに、そのサブ・ディレクトリとして、ホスト名と同じ名前のディレクトリ名の** Web **ドメインを作成します。**

```
$ createwebdomain -docroot /mnt/public_html/submerge
/webdomains/10.1.1.1/submerge.jp.oracle.com
```

今度は、もう 1 **つの** IP **アドレス**(192.168.0.1)**で、**Web **ドメインを作成します。**

```
$ createwebdomain -docroot /mnt/public_html/axis
/webdomains/192.168.0.1/axis.jp.oracle.com
```

最後に、同じ IP **アドレスで、もう** 1 **つ** Web **ドメインを作成します。**

```
$ createwebdomain -docroot /mnt/public_html/purposemaker
/webdomains/192.168.0.1/purposemaker.jp.oracle.com
```

このときの JNDI **名前空間を表したものが、図** 5-2 **になります。**

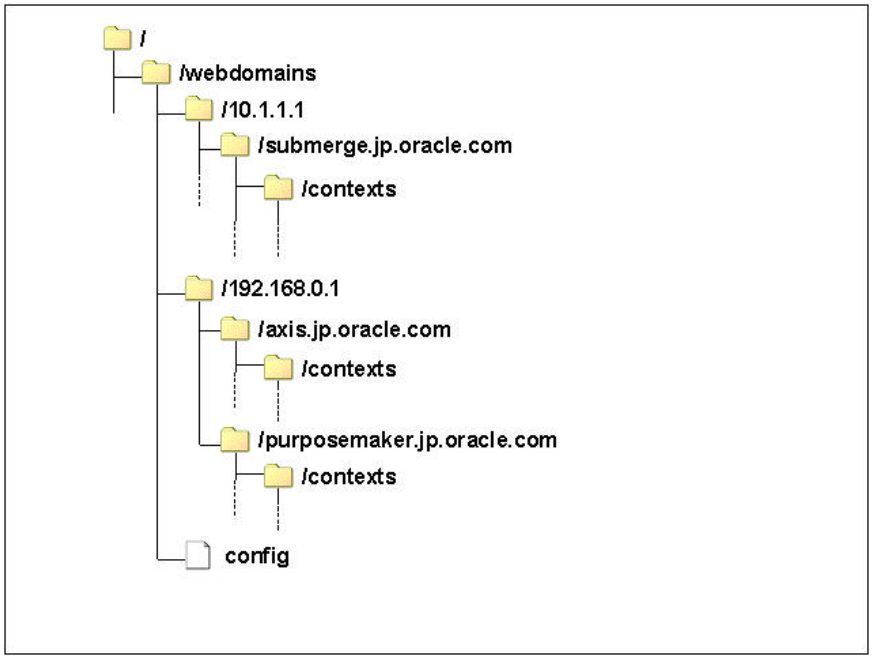


図 5-4. マルチ・ドメイン Web サービスでの JNDI 名前空間

例えば、この Web サービスのポート番号が 7700 とすると、それぞれ、ブラウザから、

http://submerge.jp.oracle.com:7700/

http://axis.jp.oracle.com:7700/

http://purposemaker.jp.oracle.com:7700/

で、アクセスできることになります。

日本オラクル株式会社