

## 第36章 Oracle应用服务器组件

本章要点：

- 分析HTTP监听层
- 分析应用服务器层
- 分析应用层

### 36.1 分析HTTP监听层

Oracle应用服务器（OAS）使用一个Web监听器等待服务请求的到来。通常，一个监听器会设定一个用作电子路标的地址，告诉所有数据包旅客“我正在营业”。当然，监听器在营业之前必须先告诉Oracle应用服务器它的地址。当OAS将这个地址注册后，可以建立一条从监听器来的路。通过这种方法，当一个客户发出一个请求时，监控器可以将信息路由到应用服务器，应用服务器会执行适当的服务（参见图36-1）。

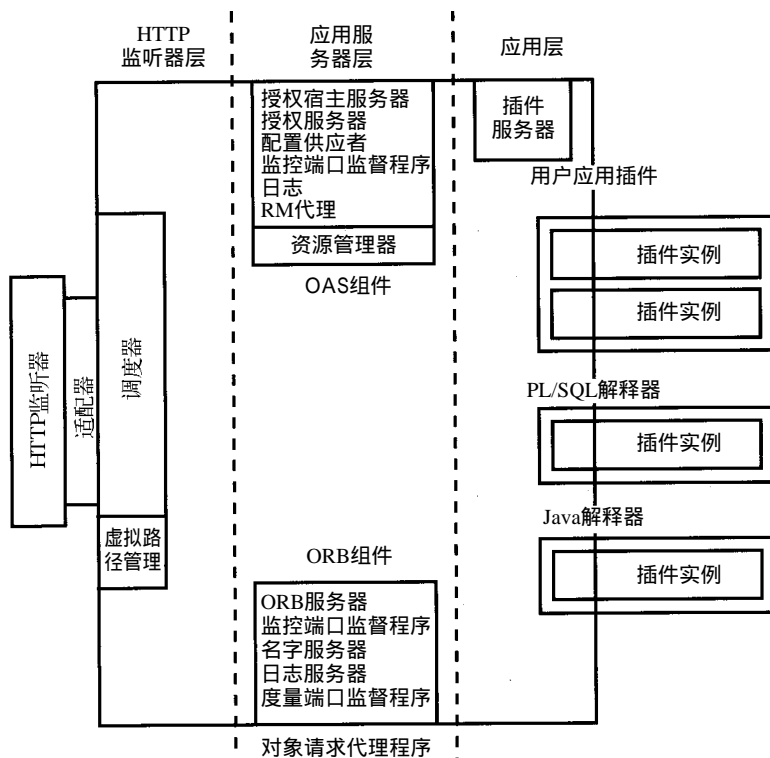


图36-1 Oracle应用服务器是由三个一起工作的概念上的层组成的

#### 36.1.1 获得更详细的细节

HTTP监听器处理一个或多个同步请求，在这种情况下，远程服务器或浏览器作为客户。

为在远程服务器上执行一个请求，必须确保对象请示代理（Object Request Broker, ORB）首先已被实例化，否则远程监听器将不存在，客户请求不能被远程执行。

幸好ORB在监控器运行期间总是处于运行状态，因为监听器不能启动，除非 ORB已运行，并且如果监听器正在运行，ORB也不能被关闭，这是因为监听器必须将它自己注册到调度器，调度器是Oracle应用服务器层的一个组件，它通过与 ORB挂钩，将监听器连接到应用服务器。

调度器的一个重要部分是虚拟路径管理器（Virtual Path Manager, VPM）。当监听器接收到一个请求时，它的形式是“虚拟路径”，即URL的后一部分。VPM负责将这个虚拟路径翻译为物理实体。如果这个物理实体是一个基于插件的应用，请求通过 ORB路由，ORB协同它的执行并将结果返回给监听器。在静态文件与 CGI脚本的情况下，请求路由回监听器，监听器创建请求自身并将结果返回给客户端。

通过结点管理器（Node Manager）创建并管理监听器，结点管理器是一个基于 Web的接口，允许管理Oracle应用服务器的所有部分。结点管理器由安装器自动启动，但也可以使用命令行工具：`owsctl start -nodemgr`稍后再启动它：

要获取结点管理器的详细信息，参见第 37章。

### 36.1.2 理解HTTP监听器的体系结构

虽然在当今的关键任务客户/服务器环境下多线程处理是必须的，但一些 IS技术人员过度使用并滥用了这项技术。他们疏忽理解的是——也很容易做到的是每个进程都需要分配内存与资源。没有适当的内存与资源分配，当打开更多的线程时，系统性能会降低，特别是当进程同步处理请求时。因而，需要在实现多线程体系结构时仔细考虑一下。

Oracle设计HTTP监听器时考虑了这个问题，监听器的任务只是接收到来的客户请求，将它们路由到应用服务器层，并将结果返回给浏览器。因而，只需要两个进程——一个监听、一个谈话。HTTP监听器也异步处理请求，意思是当它接收到一个请求时，在以后的处理中监听器不再需要这个客户。

### 36.1.3 文件的内存映射

如果文件不能永久地缓存在内存中，HTTP监听器使用一个动态内存映射模式，其中多个客户连接可以存取共同的文件。结果是，由于避免了重复硬盘读，文件存取速度加快。并且因为文件映射到内存中，操作系统可以搜索这个文件的下一个内存块，而同时将另一个文件传送到客户端。这种性能增强机制在只有一个连接的情况下仍工作得很好。

### 36.1.4 路径映射

面向对象圈中的一个重要的概念是封装或数据捆绑。在这里，不需要关心对象做了什么，只需要它按预期的方式执行。与此相似，HTTP监听器在某种意义上封装了它是怎样存储 URL路径名的。也就是说，当到达一个 Web站点时，每次都看到相同的 URL，但随着时间的推移，它可能对应不同的物理文件。OAS管理员负责正确地映射 URL路径，使用户可以看到机器文件系统上的真正的物理路径。

### 36.1.5 解析域名

域名服务器解析(Domain name server resolution)保证了客户端用户的请求发送到正确的网

络地址/URL。当这发生时，HTTP服务器记录下客户机的IP地址，通过它路由对HTML格式的请求。

域名服务（DNS）服务器用作客户的IP地址与ASCII主机名转化的参考。在这个解决方案处理中，可以包括多个主机，即在等候地址解析之前，可能要花费一段时间。

HTTP监听器提供给Oracle应用服务器系统管理员四种处理DNS任务的方法：

连接时解析IP地址。

需要时解析IP地址。

当CGI脚本或应用插件需要时解析IP地址。

从不解析IP地址。

使用DNS解析方案，这个进程的结果缓存在内存中，虽然多个连接可能在同一时间窗发生，也有可能来自同一客户机。由于性能的原因，DNS解析方案的缺省配置是从不解析IP地址。

### 36.1.6 HTTP监听器配置参数

HTTP监听器的配置参数存储在配置文件中，当监听器初始启动时，读这个文件。可以使用命令行工具启动监听器，这个工具在Oracle应用服务器的不同版本中是不同的，4.0版本也不例外。使用如下命令从命令行启动监听器：

```
owsctl start -l myListener
```

结点管理器，可以被任何窗体式Web浏览器存取，消除了手工编辑监听器配置文件的需要，在多数情况下，为每一个参数提供了解释性帮助文本。这一节文本完整地记录下配置文件中的参数，用于系统管理员想手工编辑文件的情况。

Web监听器配置文件分为多节，以括号内的节名开始，例如 [NetInfo]。使用名字 = 值对逐一配置参数，配置参数在等号左边（lvalue），值在右边（rvalue）。例如，下面就是一个典型的配置文件：

```

;
[NetInfo]
MaxConnectCount    = 338
;   The maximum number of connections. 338 is an OS limitation
;   based on the number of filehandles a single process can handle.
DNSResolution = NEVER
ServerPID = /u01/app/oracle/admin/ows/website40/httpd_csac07/myweb/myweb.pid
;
[LogParams]
TimeStyle = GMT
AdminFile = /u01/app/oracle/admin/ows/website40/httpd_csac07/myweb/myweb.err
;
[Server]
InitialFile        = wwwIndex.html
;   The "default" file accessed when no file is specified,
;   ie "http://www.csac.com/" would return
;   http://www.csac.com/wwwIndex.html. Note that while
;   wwwIndex.html is the default for OAS,
;   most sites use "index.html" as their default.
DefaultMIMETYPE    = application/octet-stream
DefaultCharset      = iso-8859-1
PreferredLanguage   = en
ImageMap            = map
UseDirIndexing      = FALSE
CGITimeout          = 900
;

```

```

[SecureInfo]
UserID          = root
GroupID         = dba
;
;   Only applicable on UNIX systems. Note that if you are
;   listening on a port under 1024, the process must run as root.
;
[DirMaps]
;Physical Directory      Flag          Virtual Path shown in URL
;
;   The flag specifies whether content is Non-CGI, CGI, or Win-CGI,
;   and whether the designation is Recursive or Non-Recursive.
/u01/app/oracle/product/8.0.4/ows/4.0/doc/      NR      /
/u01/app/oracle/product/8.0.4/ows/4.0/admin/doc/ NR      /ows-adoc/
/u01/app/oracle/product/8.0.4/ows/4.0/admin/img/ NR      /ows-aimg/
/u01/app/oracle/product/8.0.4/ows/4.0/bin/       CN      /ows-bin/
/u01/app/oracle/product/8.0.4/ows/4.0/doc/       NR      /ows-doc/
/u01/app/oracle/product/8.0.4/ows/4.0/admin/img/ NR      /ows-img/
/u01/app/oracle/product/8.0.4/ows/4.0/classes/   NR      /ows-vbclasses/
/u08/htdocs/                                     NR      /soaig/
;
en          iso-8859-1          eng
en          unicode-1-1        engU          uc
fr-CA      iso-8859-1          frc
fr-FR      iso-8859-1          fr
jp-JP      iso-2022-jp         jp
jp-JP      unicode-1-1-utf-8   jpU
;
;   Used by NLS functions.
;
[MIMETypes]
text/html   htm                html
image/jpeg  jpg                jpeg
image/gif   gif
appl/text   doc
text/plain  txt                ksh          lst
application/pdf pdf
application/postscript ps
model/vrml  wrl                vrml
video/mpeg  mpeg               mpg          mpe
;
[Encodings]
compress    Z
gzip        gz
;
[DynApps]
/u01/app/oracle/product/8.0.4/ows/4.0/lib/ndwfss.so oracle_adp_init
;
;   Directs the listener to the Adapter.
;
[MultiPort]
; IP address      TCP Port      Encryption      Host Name      Base Directory
;   LogInfo Directory      Client Authentication
ANY          80            NORM            csac07.csac.com /
➔/u01/app/oracle/admin/ows/website40/httpd_csac07/myListener/ NONE
ANY          443           SSL             csac07.csac.com /
➔/u01/app/oracle/admin/ows/website40/httpd_csac07/myListener/ NONE
;
;   A single listener can listen on multiple ports and multiple IP addresses,
;   and can direct logging for each IP/port combination to a different
➔directory.

```

### 36.1.7 第三方HTTP监听器

因为HTTP监听器与ORB对话使用调度器作为中介，所以在 Oracle HTTP监听器中没有限制。只要有适当的适配器，就可以使用任何 HTTP监听器（参见图36-2）。

在Oracle应用服务器中使用第三方 HTTP监听器，需要“注册”它。从 OAS欢迎页选择“OAS Utilities”，并展开左边的主菜单，从那里再展开“External Listener”结点及“Register”结点以找到合适的适配器。点击相应的适配器并依照下面的菜单注册你的外部监听器。

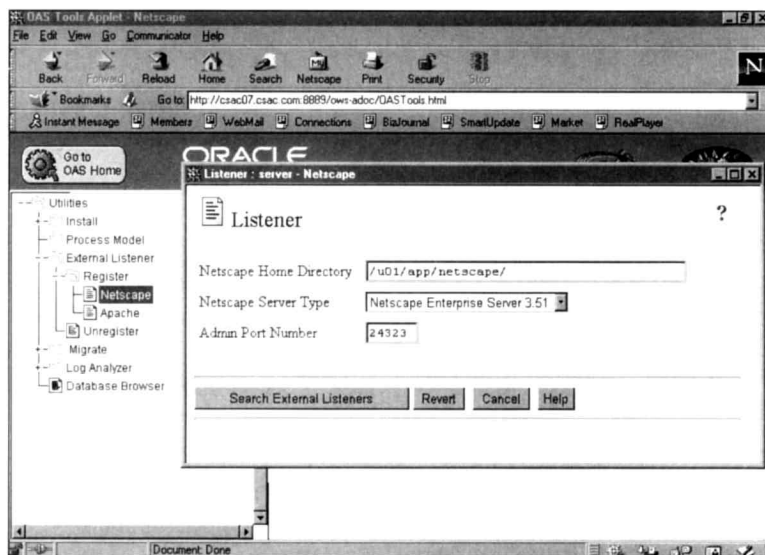


图36-2 注册一个外部监听器

注意 4.0.7版本具有由于Netscape与Apache（UNIX上）与微软的因特网信息服务器（在Windows NT上）的适配卡。

## 36.2 分析应用服务器层

HTTP成功地接收到客户端送来的信息后，将权力传递给应用服务器层。在所有的 Oracle 应用服务器组件中，应用服务器层（以前称为 Web请求代理或WRB）是最关键的，因为监听器与应用插件通过它代理他们的信息。

WRB合并了多线程与多进程体系结构，导致机器生产力的增长，几乎可以同步处理多个请求。WRB也支持异步独立进程，所以客户与服务器资源不必被迫等待返回结果集或其他信息。

当调度器通知ORB它需要一个新的插件执行实例时，资源管理器组件（Resource Manager, RM）检查所有可用的插件实例。如果没有可用的，它指示一个现有的插件服务器创建一个实例。如果所有现有的插件服务器都具有了最大数量的实例，RM指示插件制造厂创建一个新的插件服务器。如果插件服务器已到最大数，请求被排队。

注意 对象间信息通信遵循常用对象请求代理体系结构（CORBA）。在核心中，支持CORBA模型的系统具有担负以下角色的对象：客户、代理或服务器，每个都帮助从终端用户机将信息传递到本地或远程服务器。

### 36.2.1 WRB调度器

已经介绍了这么多信息，现在是更深一点钻研的时间了。在 Web监听器向WRB发送一条

信息后，WRB的调度器决定客户端请求的对象类型。这个调度器详细地查阅 WRB配置文件，这个文件帮助调度器决定虚拟路径与 WRB服务的关系。如果调度器找不到适合客户请求的关系，它将请求返回给Web监听器，这使信息生命循环进行下去。

当调度器收到基于插件应用的请求时，请求需要被路由到一个插件实例，插件实例作为插件用品的一个线程运行。如果调度器知道一个可用的插件实例，它使用 ORB向这个实例发送请求。如果调度器不知道可用的实例，它与应用服务器层联系并要求一个新的实例。参考物返回给调度器，并存储在它的对象缓存中以便当前与将来之用。

使用这种体系结构可以很自然地在服务对象实例之间动态分布进程负载，这有助于最小化网络连接与最优化 Oracle应用服务器性能。作为 Oracle应用服务器的系统管理员，当使用多进程时（或对 NT用户是服务实例），必须时刻关注潜在的资源消耗与性能退化。可以为每个应用指定最小与最大的插件服务（进程）数量，也可以指明这些内部的最小与最大插件实例（线程）数量。

注意 并不是所有的应用都可以是多线程的。例如，认为PL/SQL插件是有“疆界的”，而不是“无疆界的”，所以每个插件服务器最大只能具有一个插件实例。

在内核中，每个对象都有一个接口提供了将“产品”传递给客户的基本方法（函数）。在面向对象的术语中，这种服务职责的接受与由此引起的服务传递称为“捆绑契约”（binding contract），而且，正如一些客户/服务器专家所指出的，“对象的位置应该对客户与对象实现是透明的。”

在这种情况下，OAS与ORB模型一同定义，事实上使用因特网通信的 CORBA。通过使用 CORBA，ORB背后的主要思想是：一个对象，作为代表一些客户对象的代理，处理远程服务的请求，然后 ORB通过网络或在你的机器上提供给客户透明地调用服务对象上的对象方法的能力。这意味着不必关心提供你所需要服务的对象的操作系统或平台。这将它自己带入数据捆绑或封装及分布式安装，在分布式安装中，功能可以被扩展到许多机器或结点上。

### 36.2.2 IPC支持

与传输无关的ORB协议处理所有的进程间通信（IPC）。Oracle应用服务器支持标准IPC机制，IPC是UNIX平台上客户/服务器体系结构的核心与灵魂。然而，IPC并不是最简单的客户/服务器工具套，特别是当它用于开发多层、多进程应用时。我的直觉是 Oracle是由于在基于UNIX平台上过多的后台进程才提供这种支持的。

Oracle应用服务器 4.0版本支持IPC及其他的机制，如OMX、Oracle的遵守CORBA的ORB。此处的意见是服务应该同工业标准分布式对象一起实现，因而实现跨越多结点的分布与可伸缩性。

### 36.2.3 应用服务器层组件

这个层的组件主要分为两大类：ORB组件与应用服务器组件。

#### 1. ORB 组件

我们称 ORB为一个单独的对象，但它实际上是由共同工作的五个进程组成的（在 Windows NT中进程名以“40”结尾）。

监控端口监督程序 (mnmon40) ——启动并监控其他的ORB组件。

日志服务器 (mnlogsrv) ——日志的状态及错误信息。

公制端口监督程序 (mnorbmet) ——由WRB监控进程使用, 收集应用的统计信息。

名字服务器 (mncnmsrv) ——ORB服务器、所有的应用服务器组件、所有的监听器 (除了节点服务器), 及所有使用名字服务器注册的组件。

ORB服务器 (mnorbsrv) ——处理从分布式对象来或返回的请求。

## 2. 应用服务器组件

ORB只是方便连接。请求实际是由应用服务器组件处理的。这些进程以一至两种模式运行: 压缩模式与非压缩模式。在非压缩模式下, 所有以后的进程都是可见的, 并且可以控制。在压缩模式下, 下面的一些进程显示为“oassrv”进程。

验证服务器 (wrbasrv) 与验证主机服务器 (wrbahsrv) ——使用一个代理与服务器验证提供者, 这些进程处理监听器与插件的验证。

配置提供者 (wrbcfg) ——所有的配置信息都包含在一个单独的文件, wrb.app文件中, 即使对远方结点也是这样。配置提供者读这个文件并提供适当的信息。

日志 (wrblog) ——在一个集中的位置记录应用服务器信息、错误、警告等等, 这个集中的位置可以是一个文件或一个数据库。

监控端口监督程序 (wrbmon) ——监控应用服务器的进程与对象。

资源管理器 (wrbroker) ——只在主结点存在, 这个进程关注负载管理, 追踪插件服务器与实例的数量。

RM代理 (wrbrmpxy) ——维护指向JCORBA与EJB对象的对象。

## 36.3 分析应用层

在Oracle应用服务器上建立应用时, 有一个金字塔形的选择。可以建立 PL/SQL过程、移植Perl CGI脚本、建立Java伺服器或JCORBA对象, 甚至使用企业级Java Beans。

### 36.3.1 使用PL/SQL插件

有了PL/SQL插件, 可以使用Oracle存储过程开发自己的Web应用。Oracle向它的用户担保使用PL/SQL与数据库对象交互将比使用其他语言容易得多。显而易见的是, 由于 Oracle应用服务器使用Oracle数据库更为成熟, 对最适于移植与最具有伸缩性的问题的最可行的方法是并入PL/SQL。

在速度方面, PL/SQL插件自从第一版以来有了长足改进。这个改进来自于这样一种事实: 每个PL/SQL代理在请求到来之间的空闲中, 仍然连接到 Oracle数据库而不是在它结果返回后就与数据库断开。此处的思想是没有必要为每一个存储过程的执行建立一个新的数据库, 在孤立测试以确保不会产生不良影响之后, 应该使用 PL/SQL的最新版本。

PL/SQL代理执行两项任务, 允许存储过程以创建动态 HTML页:

它调用一个参数化的过程并传入 HTTP请求中装载的真实参数, Web应用服务器必须将一个HTTP请求翻译为PL/SQL。

它提供了一种存储过程的方法, 将数据返回给调用者, 此处低级别的细节对你无缝的。

对于非骨干程序员，这里有比所需要的更多的信息。PL/SQL插件的主要优点在于它隐藏了与数据库交互的复杂性。基本上，当它接收到一个请求时，插件服务器将其记录到数据库日志中，执行一个存储过程并向浏览器返回任何结果。

除了执行这些工作以外，OAS还包括PL/SQL工具箱，它是封装的过程、函数与工具的集合，用以实现生产力与确保合法的HTML输出。这个工具箱包括的PL/SQL版本有全部HTML3.2的标签（在http包中），及帮助执行更高级任务的工具。

要记住的最重要的事情是从浏览器调用一个存储过程与从如SQL\*Plus工具调用该过程是一样的。任何插入、更新、删除等等，都以相同的方式执行，只有一个重要的区别：除非正使用事务，否则只要请求一结束，插件实例将退出数据库，修改将被提交（见清单36-1）。

清单36-1 示例PL/SQL过程

```
procedure demo_http (the_user varchar2) is
  user_host      varchar2(150);
  cnt            integer;
begin
  http.htmlOpen;
  http.headOpen;
  http.title(the_user||'s Page');
  -- Takes the value of "the_user" from the input parameters
  http.headClose;
  http.bodyOpen;

  user_host := owa_util.get_cgi_env('REMOTE_HOST');
  -- Gets the value of the "remote host" CGI environment variable. You
  -- can use any environment variable

  insert into user_history (web_user, web_host, web_time)
    values (the_user, user_host, sysdate);
  -- This will be inserted as if the procedure was executed from SQL*Plus

  http.p('Good morning, '||the_user||'!<P>');
  -- http.p() can be used to out put any arbitrary text to the browser,

  -- including HTML tags

  select count(*) into cnt from user_history;
  http.p('There have been '||cnt||' visits to this page so far.');
```

— You can easily integrate information from the database into your pages.  
— This could just as easily have been names, times, anything stored in the  
— database.

```
  http.bodyClose;
  http.htmlClose;

end;
```

从浏览器执行这个过程，只需将浏览器设为：

http://www.yourhost.com/myApp/plsql/demo\_http?the\_user=Nick

假设已将虚拟路径“myApp/plsql”指派为指向PL/SQL插件。

为帮助了解这是如何工作的，可以如下方式进行测验：注册进入SQL\*Plus，在调用这个网页之前与之后查看表中的内容，然后向表中手工插入一行，提交这个修改，再次调用这个网页。

不能对参数使用定位符号，名字必须确切匹配，每个进入的参数都是 `varchar2` 类型，因为它是 URL 文本的一部分。聪明的方法是，如果在包中重载过程，他们可以具有相同的参数类型，虽然参数的名字是不同的。

提示 可以在 SQL\*Plus 中使用下述语句检查过程的输出：

```
set serveroutput on
exec myprocedure;
exec owa_util.showpage
```

Oracle 包括了成打的应用工具以简化工作，在开始工作之前仔细查看工具箱包的文档。

注意 Oracle 实现了随时支取的 JAVA 存储过程，就像它们是 PL/SQL 存储过程一样，所以它们可以同样的方式从浏览器随时支取。

对于核心程序员，Oracle 继续支持 C 编程语言，将它合并入一个真正的插件，称为 UNIX 上的 C Web 插件，并简化了 Windows NT 上的 C 插件。

### 36.3.2 WRB SDK 的孩子：C Web 插件

在 2.0 版本中，Oracle 引入了 Web 请示代理软件开发者的工具，或 WRB SDK，这个开放式的 API 鼓励用户与解决方案提供者扩展 Oracle 预想的服务器，他们建立的应用称为“插件”，但现在简称为“应用”。这些应用是用 C 写的，可以做任何 C 程序可以做的工作，如与付款系统集成。他们作为共享库运行，在运行时动态加载并放入缓存。

如果为以前的版本书写 C 插件，需要使用结点管理器中的 OAS 工具将其升级，但它们仍可工作。老的 API 仍被支持，但在将来的版本中可能被撤走。

### 36.3.3 WRB 应用程序界面

可以使用 WRB API 与 ORB 注册三个基本回调函数：

初始化。

请求处理。

关闭。

这些方法依据典型的类结构集合到一起，可以初始化对象、在对象初始化后发送与接收信息、使用完成后破坏这个对象。这个结构使它自己非常易于面向对象方法的系统开发。

WRB API 含有的辅助方法与函数极多。这些方法中的一些将在本章稍后讨论。这些方法将从先前提到的三个基本回调直接调用。

Oracle 对产品开发的开放途径指的是用户与方案解决提供者 / 合作者联合开发 WRB API。这个合作努力意味着 API 将在 Oracle 应用服务器与第三方应用之间提供最大的开放与合作能力。

### 36.3.4 理解插件与 ICX

ICX，代表插件之间的交换，是一个 Oracle 应用服务器支持的机制，推动插件之间的通信，它允许不同的机器之间进行通信。特别是使用基于 Java 的插件。

#### 1. WRB 插件

每个插件都遵循一个体系结构 / 模型类型，插件开发者只需要知道如下三种设计模型：

请求-响应模型。

会话。

事务。

## 2. 请求-响应模型

与膝跳反射几乎相似，WRB插件实例处理单独的客户请求并对每一个预期的响应执行一个请求。预期的请求被分派后，插件实例具有了“健忘症”，此时没有数据与客户端相关，请求也不是永久地到处存储。如果熟悉缺少规则HTTP请求追踪的状态信息，这将是非常常见的。所以，插件的Exec方法应该将请求作为变元接收，并返回相关信息而不需追踪任何状态数据。

## 3. 会话模型

在客户与一个特定的执行中心插件实例之间建立会话。然后插件使用会话机制维护这两个之间的持久关联。可以通过设置休息时间确定连接坚持多长时间。然后插件的Exec方法将声明与客户相关的信息并初始化请求。Exec方法将这个声明信息存储在拥有实例插件的应用的上下文结构中，即，应用负责创建插件的一个实例（初始化插件实例，设计时插件的运行状态对象拷贝）。

注意 插件运行时的体系结构与设计时体系结构的区别的详细信息，参见Grady Booch、Jim Rumbaugh或Lvar Jacobson在面向对象方法学方面的著作。

会话机制确保在客户端与可执行插件实例之间有一个关联，这个关联提供给插件在它自己与客户端之间的一个特殊的焦点。这个实例特殊化正是运行时与设计时体系结构不同的意义所在。在任何给定的时间，可以执行一个插件的许多实例，每个实例含有特定用户的特定状态与数据。也就是说，如果你有一个为特定的银行分支管理许多经常帐户的插件，插件的一个实例具有\$5 000的结余，另一个实例具有\$230.98的结余，另一个具有\$1 209的结余，等等，所有的都应在本日末协调他们的批量执行。数据完整性不会被破坏，因为每个插件实例只关心与一个帐户相关的信息，然后可以保存与应用中每个帐户有关的状态信息，将其放入文本结构，以便此信息在当前用户会话期间是存活的。

注意 在3.0版本中，每个插件实例在它自己的进程空间运行。在4.0版本中，实例可以共享进程空间（在插件服务器进程中）。

## 4. 事务模型

如果在分析与设计中，意识到插件在管理数据库事务时需要集中，事务模型正合适，Exec方法将承担事务处理的角色，在这种模型的实现中包括一些常用状态：

插件接收一个请求。

插件开始一个事务。

插件在事务范围内执行增加的更新。

如果每件事都顺利，插件提交一个事务。

如果不是每件事都顺利，插件进行事务上的回滚（事务的状态：结束）。

插件向客户端发送一个响应（认可或错误）。

在一个调用中不必产生事务（调用Exec方法）。一些事务可以横跨许多对插件的Exec方法的调用。在此是插件的Exec方法必须决定对当前场景应用哪种前述的状态。如果设计将来调用这个特性，这个模型允许支持与任何事务无关的查询（更详细的信息，参见插件设计方面

的WRB事务服务API指南)。

如果你已充分理解,并购买 Oracle应用服务器的企业版,你可以从事务服务库中得到更多的API帮助(在Oracle应用服务器的基本版中不包括这些服务)。通过这个库,你或你的开发小组可以从你的插件执行任何数量的数据库事务。Oracle将WRB事务服务建立在TX接口之上,与它被x/Open公司定义了一样。

注意 如果需要关于这个有帮助的特性的更多的信息,查看X/开放的分布式事务处理:TX(事务分界)规范。

通过高级事务服务,插件可以执行跨越多个HTTP请求的数据库事务。对数据库的真正存取通过基于数据库存取的API被建立,如OCI或ProC,作为事务服务的补充。事务函数如commit与rollback并不通过这些服务完成。为使插件可以使用事务服务,必须启用这个插件的事务服务。查阅事务服务文档以获取更详细的细节。

### 5. 看看ICX

WRB插件之间交换服务API允许一个WRB插件将HTTP请求发给另一个WRB插件。

使用ICX帮助系统体系结构设计提供一个应用或企业领域特殊行为的插件。ICX允许插件旋转在不同的机器上,因而改进了Oracle应用服务器的性能,并对网络提供了有价值的负载均衡。这也意味着增长的存储空间(所有应用的组件并不位于一台机器上),及更灵活的体系结构。与好的类与对象模型结合,ICX使维护许多交互式的插件相对容易一些。

表36-1显示了WRB插件之间交换服务API中的所有方法。

表36-1 ICX APE方法

方 法	描 述
WRB_ICXcreateRequest()	创建一个请求对象
WRB_ICXdestroyRequest()	摧毁一个请求对象
WRB_ICXfetchMoreData()	当WRB_ICXmakeRequest()返回WRB_MOREDATA结构时,得到更多的数据
WRB_ICXgetHeaderVal()	得到特定响应头的值
WRB_ICXgetInfo()	得到有关一个请求的信息
WRB_ICXgetParsedHeader()	得到响应头
WRB_ICXmakeRequest()	执行一个请求
WRB_ICXsetAuthInfo()	为一个特定请求设置授权头
WRB_ICXsetContent()	为一个特定请求设置内容数据
WRB_ICXsetHeader()	为一个特定请求设置头
WRB_ICXsetMethod()	设置一个特定请求中使用的HTTP方法
WRB_ICXsetNoProxy()	指明代理服务器不能使用的域
WRB_ICXsetProxy()	指明一个代理服务器
WRB_ICXsetWalletInfo()	设置ICX请求去使用SSL证明用于授权

ICX API还支持两个枚举的类型:

WRBInfoType

WRBMethod

下面的部分描述每一种方法。

WRB\_ICXcreateRequest() 这个方法分配并返回对一个不透明请求对象的处理,对一个由给定URL指明的请求编码。程序清单 36-2显示了创建一个请求对象所需要的代码:

清单36-2 用WRB\_ICXcreateRequest()创建一个请求对象

---

```
dvoid *
WRB_ICXcreateRequest(void *WRBCTX,
text *url);
Parameters
Return Values
WRB_ICXcreateRequest() returns a handle to the newly created request object.
WRB_ICXcreateRequest() returns NULL on failure.
```

---

除了创建请求对象，或许还想发出请求，发出请求的过程如下：

1) 调用WRB\_ICXcreateRequest ()。

2) 调用WRB\_ICXmakeRequest ()。

3) 发出请求。

4) 如果出现了错误，通过调用 WRB\_ICXdestroyRequest ()方法终止请求，这个调用释放WRB为请求分配的资源。

\* WRBCTX是指向 WRB应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，text\*url是URL请求的指针。

WRB\_ICXdestroyRequest () 这个方法释放WRB为特定请求分配的资源。释放不再使用的对象占用的资源是一个很好的编程习惯。程序清单 36-3展示了在使用完请求对象后，如何摧毁它。

清单36-3 使用WRB\_ICXdestroyRequest ()摧毁一个请求对象

---

```
WAPIReturnCode
WRB_ICXdestroyRequest(void *WRBCTX,
dvoid *hRequest);
Parameters
Return Values
WRB_ICXdestroyRequest() returns a value of type WAPIReturnCode.
```

---

请求完成后，必须调用WRB\_ICXdestroyRequest ()，只要不使用WRB\_ICXmake Request ()发出一个请求，就可以使用 WRB\_ICXdestroyRequest ()取消一个由WRB\_ICXcreateRequest ()创建的请求。

\* WRBCTX是指向 WRB应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，\*hRequest指明将被摧毁的请求，WRB\_ICXcreateRequest ()将返回这个控制。

WRB\_ICXfetchMoreData () 当一个先前对 WRB\_ICXmakeRequest ()的调用返回WRB\_MOREDATA时，WRB\_ICXfetchMoreData ()重新得到请求数量的字节。如果这个请求的数量比可以获得的大，这个方法返回可以获得的字节数量。程序清单 36-4详细展示了这个方法。

清单36-4 使用WRB\_ICXfetchMoreData ()获取更多数据

---

```
WRBAPIReturnCode
WRB_ICXfetchMoreData(dvoid *WRBCTX,
dvoid *hRequest,
dvoid **response,
ub4 *responseLength,
ub4 chunkSize);
Parameters
```

---

**Return Values**

WRB\_ICXfetchMoreData() returns a value of type WAPIReturnCode.

\*WRBCTX是指向 WRB 应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，.dvoid \*hRequest指明将被摧毁的请求，dvoid \*\*response是一个指针，指向的位置是方法存储响应数据的指针，ub4 \* responseLength是指向函数存储响应数据长度（以字节为单位）的指针。如果ub4 chunkSize具有一个非零值，需要重复调用 WRB\_ICXfetchMoreData ()方法直至接收到整个响应。如果ub4 chunkSize的值为0，不返回数据。

如果对WRB\_ICXmakeRequest ()的调用返回WRB\_MOREDATA，可以调用这个方法必要多的次数以获取请求响应 chunkSize额外的字节，需要这样做直至接收到所有的响应数据。

WRB\_ICXgetHeaderVal () 在使用这个方法之前，必须调用 WRB\_ICXmakeRequest ()方法，它发出一个请求。WRB\_ICXgetHeaderVal ()使用这个请求的响应返回一个指定 HTTP头的值（参见清单 36-5）。

清单36-5 使用WRB\_ICXgetHeaderVal ()获取一个指定响应头的值

```
text *
WRB_ICXgetHeaderVal(void *WRBCTX,
dvoid *hRequest,
text *name);
Parameters
Return Values
WRB_ICXgetHeaderVal() returns the value of the specified header.
WRB_ICXgetHeaderVal() returns NULL on failure.
```

WRB\_ICXgetHeaderVal ()返回的指针指向由 WRB\_ICXmakeRequest ()发出请求的响应的特定HTTP头的值，这个指针对于检索响应头含有的响应数据是非常有价值的。

\*WRBCTX是指向 WRB 应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，\*hRequest指明WRB用以提取响应头值的响应，text \* name是指向你想要的头的名字的指针。

WRB\_ICXgetInfo () 这个方法返回含有有关特定请求信息的字符串，可以指定想要返回的信息的种类，程序清单 36-6详细展示这个方法。

清单36-6 使用WRB\_ICXgetInfo ()获取有关一个请求的信息。

```
text *
WRB_ICXgetInfo(void *WRBCTX,
dvoid *hRequest,
WRBInfoType infoType);
Parameters
Return Values
WRB_ICXgetInfo() returns a pointer to the requested information as a character
string. WRB_ICXgetInfo() returns NULL on failure.
```

当一个ICX请求完成时，调用 WRB\_ICXgetInfo ()以获取有关这个请求的信息。触发这个完成状态的事件是 WRB\_ICXmakeRequest ()的完成与返回。此处无用的是万一 WRB重新以适当的授权信息执行请求时，你会获得界名 (realm name)。

\*WRBCTX是指向 WRB 应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，dvoid \*hRequest指明你需要的是哪个请求的信息，infoType与WRBInfoType代表指明你想要的信息类型的代码，infoType参数指明返回信息的种类。

WRB\_ICXgetInfo ()接受将由WRBInfoType指明的请求信息类型的变元返回给调用者，程序清单36-7显示了这种类型的结构。

清单36-7 WRBInfoType枚举类型

---

```
typedef enum _WRBInfoType
{
    STATUSCODE,
    HTTPVERSION,
    REASONPHRASE,
    REALM
} WRBInfoType;
WRBInfoType Values
```

---

STATUSCODE代表HTTP响应代码，HTTPVERSION代表响应使用的HTTP协议的版本，REASONPHRASE代表与HTTP响应代码一致的原因文本串，REALM代表在响应中指明的授权界名。

WRB\_ICXgetParsedHeader () 在使用这个方法之前，必须调用WRB\_ICXmakeRequest ()发出一个请求，WRB\_ICXgetParsedHeader ()使用这个请求的响应返回指定头的值，程序清单36-8展示了这个方法。

清单36-8 使用WRB\_ICXgetParsedHeader ()获取响应头

---

```
WAPIReturnCode
WRB_ICXgetParsedHeader(void *WRBCTX,
    dvoid *hRequest,
    WRBpBlock *hPblock);
Parameters
Return Values
WRB_ICXgetParsedHeader() returns a value of type WAPIReturnCode.
```

---

这个方法返回由hPblock指向的地址，这个地址是一个含有由WRB\_ICXmakeRequest ()发出的ICX请求响应的参数块。此处无用的是你可以检索响应头包含的响应数据。

\*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针，dvoid \*hRequest指明请求，WRBpBlock \* hPblock是指向函数用以存储含有解析头数据的参数块的位置的指针。

WRB\_ICXmakeRequest () 这个方法发出指定的请示，程序清单36-9展示了这个方法。

清单36-9 使用WRB\_ICXmakeRequest ()发出一个请求

---

```
WAPIReturnCode
WRB_ICXmakeRequest(void *WRBCTX,
    dvoid *hRequest,
    void **response,
    ub4 *responseLength,
    ub4 chunkSize,
    ub1 sendToBrowser);
Return Values
WRB_ICXmakeRequest() returns a value of type WAPIReturnCode.
```

---

在调用WRB\_ICXcreateRequest ()创建一个请求及其他的ICX API函数如WRB\_ICXsetHeader ()与WRB\_ICXsetContent ()准备请求后，可以调用WRB\_ICXmakeRequest ()发出请求。

\*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针，

dvoid \*hRequest指明要发出的请求。

Void \*\* response是用以指向函数存储指向响应数据的位置的指针，ub4 \* responseLength是指向函数用以存储响应数据字节长度的地址的指针。

ChunkSize ub4有一个非零值，请求响应的字节长度必须与这个值匹配。在这种情况下，需要反复调用WRB\_ICXfetch MoreData ()方法直至接收到全部响应。如果ub4 ChunkSize的值为0，不返回数据。如果ub1 sendToBrowser是非零值，WRB将直接将响应发送给起始浏览器。在这种情况下，响应参数返回时将为空。

WRB\_ICXsetAuthInfo () 这个方法设置伴随指定请求的授权头数据（参见清单 36-10）。

清单36-10 使用WRB\_ICXsetAuthInfo ()设定指定请求的授权头

---

```
WAPIReturnCode
WRB_ICXsetAuthInfo(void *WRBCTX,
dvoid *hRequest,
text *username,
text *password,
text *realm);
Parameters
Return Values
WRB_ICXsetAuthMethod() returns a value of type WAPIReturnCode.
```

---

如果你的插件向另一个插件发出请求，反过来需要你的插件为它自己授权，你需要调用WRB\_ICXsetAuthInfo ()方法，这样做为其他插件的每一个请求设置授权头。

\*WRBCTX是指向WRB应用引擎传递给你的插件方法不透明的WRB内容对象的指针，dvoid \*hRequest指明为哪个请求建立授权。WRB\_ICXcreateRequest ()返回这个句柄，text \* username是指向请求授权的用户名的指针。用户名必须在指定的界中定义。text \* password指向用户名口令，text \* realm指向定义用户名的授权界的名字。

WRB\_ICXsetContent () WRB\_ICXsetContent ()设置指定请求的请求内容（参见清单 36-11）。

清单36-11 使用WRB\_ICXsetContent ()为一个指定请求设置内容数据

---

```
WAPIReturnCode
WRB_ICXsetContent(void *WRBCTX,
dvoid *hRequest,
WRBpBlock hPBlock);
Parameters
Return Values
WRB_ICXsetContent() returns a value of type WAPIReturnCode.
```

---

为一个特定查询建立内容数据，执行以下步骤：

- 1) 调用WRB\_createPBlock ()分配一个含有内容的参数块。
- 2) 将参数块传递给WRB\_ICXsetContent ()，通过传递WRB\_ICXcreateRequest ()返回的请求句柄指定请求。
- 3) 设置内容数据。

\*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针，dvoid \*hRequest指明是指定哪个请求的内容，这应该是由WRB\_ICXcreateRequest ()返回的句柄，WRBpBlock hPBlock描述含有请求内容的参数块。

WRB\_ICXsetHeader () WRB\_ICXsetHeader ()负责设置指定请求的HTTP头数据 (参见清单36-12)。

清单36-12 使用WRB\_ICXsetHeader ()为指定请求设置头

---

```
WAPIReturnCode
WRB_ICXsetHeader(void *WRBctx,
dvoid *hRequest,
WRBpBlock hpBlock,
boolean useOldHdr);
Parameters
Return Values
WRB_ICXsetHeader() returns a value of type WAPIReturnCode.
```

---

调用这个方法需要首先进行一些特定的调用,调用顺序如下:

- 1) 调用WRB\_createPBlock ()分配一个参数块及包含头数据。
- 2) 将参数块传递给WRB\_ICXsetHeader (),通过传递WRB\_ICXcreateRequest ()返回的请求句柄指定请求。
- 3) 为请求设置头数据。

\*WRBctx是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针,dvoid \*hRequest指明设定哪个请求的头数据。WRB\_ICXcreateRequest ()返回dvoid \*hRequest句柄,WRBpBlock \*hpBlock描述含有头信息的参数块。如果boolean useOldHdr设为TRUE,ICX除参数块中定义的数据外,还从初始请求查询相应的头数据。如果boolean useOldHdr设为FALSE,只使用参数块中的头数据。

WRB\_ICXsetMethod () WRB\_ICXsetMethod ()负责为指定的请求设置请求方法,如GET或POST (参见清单36-13)。

清单36-13 使用WRB\_ICXsetMethod ()设置指定请求中使用的HTTP方法

---

```
WAPIReturnCode
WRB_ICXsetMethod(void *WRBctx,
dvoid *hRequest,
WRBMethod method);
Parameters
Return Values
WRB_ICXsetMethod() returns a value of type WAPIReturnCode.
```

---

此处假设已调用WRB\_ICXcreateRequest ()方法创建了一个请求。调用WRB\_ICXsetMethod ()指定这个请求的请求方法,缺省请求方法是GET。

\*WRBctx 是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针,dvoid \*hRequest 指明请求的设置方法。WRB\_ICXcreateRequest ()返回这个句柄。

WRB\_ICXsetMethod ()具有一个WRBMethod类型的变元,描述请求中将使用的请求方法 (参见清单36-14)。

清单36-14 WRBMethod枚举类型

---

```
typedef enum _WRBMethod
{
OPTIONS,
GET,
HEAD,
```

---

```
POST,
PUT,
DELETE,
TRACE
} WRBMethod;
```

要获取这个枚举类型的详细信息，参考 Web 请求代理的有关文档。

WRB\_ICXsetNoProxy () 这个方法建立了代理服务器（由 WRB\_ICXsetProxy() 指定）不能使用的 DNS 域的列表，这确保了拒绝任何给定代理服务器最初试图请求的 URL，程序清单 36-15 展示了这个方法是怎样实现的。

清单 36-15 使用 WRB\_ICXsetNoProxy () 指定代理服务器不能使用的域

```
WAPIReturnCode
WRB_ICXsetNoProxy(void *WRBCTX,
text *noProxy);
Parameters
Return Values
WRB_ICXsetNoProxy() returns a value of type WAPIReturnCode.
```

如果插件调用 WRB\_ICXsetProxy () 启动代理服务器请求翻译，但你不希望所有 DNS 域都使用代理服务器到达，使用 WRB\_ICXsetNoProxy () 指定一个以逗号分隔的域的列表，请求直接发送到这些域，而不需要代理服务器的插入。

\*WRBCTX 是指向 WRB 应用引擎传递给你的插件方法的不透明的 WRB 内容对象的指针，\*noProxy 指向以逗号分隔的直接发送的请求的 DNS 域的列表。

WRB\_ICXsetProxy() 这个方法告诉插件在建立将来的 ICX 请求时使用哪个必须在防火墙外路由的代理服务器。程序清单 36-16 展示了怎样指定一个代理服务器。

清单 36-16 使用 WRB\_ICXsetProxy 指定代理服务器

```
WAPIReturnCode
WRB_ICXsetProxy(void *WRBCTX,
text *proxyAddress);
Parameters
Return Values
WRB_ICXsetProxy() returns a value of type WAPIReturnCode.
```

当基于内部网的插件需要将 ICX 请求分派到防火墙外的服务器时，调用这个方法很有用。插件将指向这个方法设定的地址。

\*WRBCTX 是指向 WRB 应用引擎传递给你的插件方法的不透明的 WRB 内容对象的指针，\*proxyAddress 以字符串形式表示代理地址。

### 36.3.5 WRB 记录器应用程序界面

在 Oracle 应用服务器与 Oracle 数据库紧密结合的前提下，应该会更熟悉与数据库连接相关的 WRB API，WRB 记录器 API 函数提供了这种例程：

WRB\_LOGopen ()——打开一个文件或建立一个数据库连接。

WRB\_LOGwriteMessage ()——将系统信息写入永久存储。永久存储可以是数据库或一个普通的文件，其中日志数据在服务器应用生命期外仍可存活。

注意 在 Oracle 应用服务器的早期版本中支持 WRBLogMessage ()。在 Oracle 应用服务器

4.0版本中，应该使用WRB\_LOGwriteMessage ()代替它。

WRB\_LOGwriteAttribute ()——将客户端定义的属性写入永久存储。

WRB\_LOGclose ()——关闭一个文件或断开数据库连接。

程序清单 36-17展示了WRB\_LOGopen ()方法的实现。

清单36-17 WRB\_LOGopen ()的语法

---

```
WAPIReturnCode WRB_LOGopen( dvoid *WRBctx,
ub4 *logHdl,
WRBLogType type,
WRBLogDestType dest,
text *MyFileName );
Parameters
Return Values
[Returns WAPIReturnCode]
```

---

从插件组件调用这个方法用以打开一个文件或初始化一个数据库连接。 WRB应用引擎将WRBctx指针传递给你的插件中的调用方法； WRBctx指向WRB内容对象； LogHdl是指向连接对象中含有的对象类型的处理：文件或数据； type指出日志条目的类型； dest指明将事务日志写入一个平面文件还是数据库； MyFileName是文件的字符串名，当将日志信息写入数据库时，它也可以为空。

程序清单 36-18中的方法将系统信息写入由 LogHdl 指明的永久存储中。

清单36-18 WRB\_LOGwriteMessage ()的语法

---

```
WAPIReturnCode WRB_LOGwriteMessage( dvoid *WRBctx,
ub4 logHdl,
text *component,
text *msg,
sb4 severity);
Parameters
Return Values
Returns WAPIReturnCode.
```

---

从插件组件调用这个方法用以将系统信息写入由 LogHdl 中的值指明的永久存储中。 WRB应用引擎将 WRBctx 指针传递给你的插件中的调用方法； WRBctx 指向 WRB 内容对象； LogHdl 是指向连接对象中含有的对象类型的处理：文件或数据库； type 指出日志条目的类型，它可以是客户端属性或信息组件，指向文本描述确定插件的类型，如 “ java ”。 msg 就是你想要记录的文本，信息不能超过 2K 长度；反引号 ( ) 用于分隔，如果在你的信息中需要含有这个字符，必须以反斜线为标记 ( 即 \ ) ； severity 只是消息的严谨级。

程序清单 36-19 中的代码将客户端定义的属性写入由 LogHdl 指明的存储中。

清单36-19 WRB\_LOGwriteAttribute ()的语法

---

```
WAPIReturnCode WRB_LOGwriteAttribute( dvoid *WRBctx,
ub4 logHdl,
text *component,
text *name,
text *value);
Parameters
Return Values
Returns WAPIReturnCode.
```

---

从插件组件调用这个方法用以将客户端定义的属性写入由 LogHdl 中的值指明的永久存储中。存储的信息对查看你的客户插件是怎样响应系统的是非常有用的。当开发组件时，追踪错误与例外处理是非常关键的，WRB 应用引擎将 WRBCtx 指针传递给你的插件中的调用方法；WRBCtx 指向 WRB 内容对象，LogHdl 是指向连接对象中含有的对象类型的处理：文件或数据库；type 指出日志条目的类型，它可以是客户端属性或信息组件，指向文本描述确定插件的类型，如“ODBC”。msg 就是你想要记录的文本；name 是一个文本项，指明你想要用日志记录下来的特定属性；value 提供附加的文本以限定你命名的属性。参见程序清单 36-20，查看 WRB\_LOGclose () 方法的语法。

清单 36-20 WRB\_LOGclose () 的语法

---

```
WAPIReturnCode WRB_LOGclose( dvoid *WRBCtx, ub4 logHdl);  
Parameters  
Return Values  
Returns WAPIReturnCode.
```

---

依据 LogHdl 中指明的值，从插件组件调用这个方法用以关闭一个文件或断开一个数据库连接。

WRB 应用引擎将 WRBCtx 指针传递给你的插件中的调用方法，WRBCtx 指向 WRB 内容对象；LogHdl 是指向连接对象中含有的对象类型的处理：文件或数据库，这个处理在 WRB\_LOGopen () 方法中创建。

### 36.3.6 使用 JWeb 插件

JWeb 插件，以前称为 Java 插件，也许会有一些混淆。JWeb 插件与 Java applets 毫无关系，Java applets 是一个下载的小程序，运行在浏览器中。本质上是一个 Java 虚拟机，JWeb 插件在服务器上运行一个 Java 应用，并直接将 HTML 返回给浏览器。浏览器不需要特殊的能力，因为它除了解释 HTML，不需要做任何事情。

Oracle 还提供了 pl2java，一个允许从 Java 应用调用 PL/SQL 的工具，及 Java Toolkit，允许将 PL/SQL 与 WRB 函数写入 Java 应用。

#### 1. 增加一个 java 应用

在运行一个 Java 应用之前，需要让 OAS 知道它在哪里，它是什么。

- 1) 访问结点管理器。
- 2) 点击 OAS 管理器。
- 3) 扩展左边的主菜单并点击应用。
- 4) 在右边，通过点击 +，增加一个新的应用。
- 5) 在应用类型下面，选择 JWeb，确认单选按键被手工选中，点击应用按钮（参见图 36-3）。
- 6) 填入应用名（即，Hello World）与显示名（helloworld），这些是你的应用将分别在应用列表与菜单中显示的。选取合适的版本号，如 1.0，为这个应用按钮点击增加插件。
- 7) 输入插件名与显示名，这些将被用于在结点管理器中显示信息。
- 8) 输入将要存放已编译的类文件的路径位置及真实路径，如 /java/myTest，并点击应用，这个路径必须是 CLASSPATH 环境变量的一部分。
- 9) 重新装载应用服务器，点击左边的 Oracle 应用服务器，然后选取“所有”与“重载”，

它看起来像一个VCR反复播放按钮。

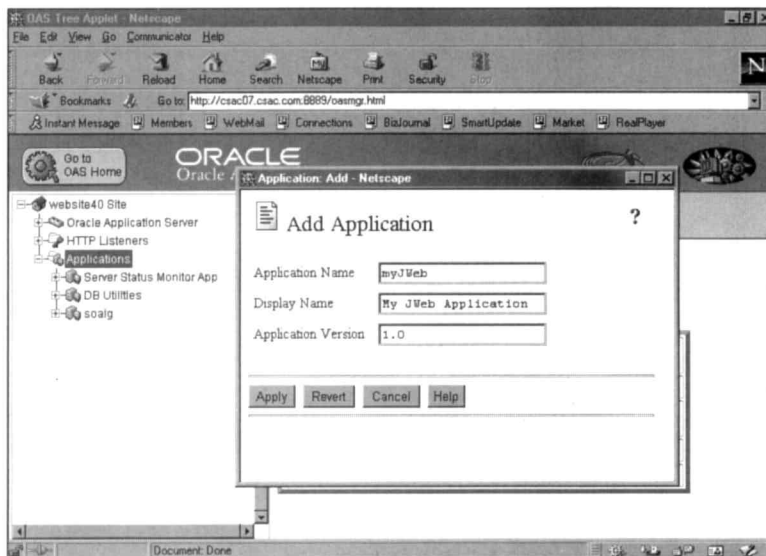


图36-3 增加一个JWeb应用

现已使用一个插件创建了“Hello World”应用。

提示 如果已有一个用PL/SQL书写的的应用，不用感觉有责任将它们转换为Java。Oracle已经构建对Java的支持，所以可以从一个程序调用另一个，不用重写PL/SQL，只需要从新的Java代码中调用它即可。

## 2. 在Java中建立一个网页

现在准备创建真正的Java应用。在此与常用的Java编程有微小的区别：你需要有一个或多个类，但用户基本上使用main()方法启动应用，并一起使用它直至完成。在这个时间内，用户看到的東西可能被许多不同的方法控制着。当为OAS创建Java应用时，记住每个单独的网页都有它自己的类，OAS将总是被main方法调用以显示它。让我们看一个样例应用（见清单3-21）：

清单36-21 Jweb应用示例

```
import oracle.html.*; // This is the Java Toolkit
class showJavaPage {
    public static void main (String args[]) {
        HtmlHead h = new HtmlHead("My Java App");
        // Just creates the HTML for the <HEAD> and <TITLE> tags
        HtmlBody b = new HtmlBody();
        b.addItem("This text was generated by my Java Application.");
        // Add text to the body
        HtmlPage p = new HtmlPage(h, b);
        // Creates the page object and adds the head and body objects
        p.printHeader();
        // All pages must have HTTP headers.
        p.print();
        // The page prints itself.
    } // main
} // class
```

在应用服务器可以获取它之前，必须将它编译为 \*.class 文件。JWeb 插件使用 Java 1.1.4，而不是 1.2 或 1.1.6，所以仔细察看你增加了什么特性。Oracle 建议选择一个使用相同版本的编译器。Oracle 也有它自己的 Java 工具，JDeveloper。

在类被编译并测试后，在创建应用时将它拷贝到你指向的路径。为在你的浏览器中察看它，到下述 URL：

<http://www.yourmachine.com/java/myTest/showJavaPage>

注意 [www.yourmachine.com](http://www.yourmachine.com) 是你的机器，/java/myTest 是你创建应用时进入的虚拟路径，注意即使你的应用称为 “Hello World”，但你实际并不能看到这个文本。应用的名字是一个任意的选择，只有应用的系统管理员使用结点管理器时需要它。

使用 JWeb 插件，你将获取与使用 CWeb 插件相同的 WRB 服务，它们包含在 Oracle.\* 包中。

提示 在版本 4.0 中，不赞成使用 Oracle.owas.wrb.services.logger 包，使用 Oracle.OAS.Services.Logger 代替。

对拿不定主意的人，LiveHTML 只是服务器端包括的 Oracle 的名字，或 SSI。SSI 是 Web 在网页上提供动态信息的最早的方法之一。它允许开发者创建一个静态 HTML 页，但在内容中“内嵌”动态信息，如日期或另一个 HTML 页。当服务器接收到对这些页的请求时，它分析这个页，增加附加的信息，并将整个事件送回给浏览器。查看 HTML 源文件没有给出这个发生的任何指示。LiveHTML 可以分为两类，第一类是 SSI。

在 SSI 中，命令以如下语法被内嵌：

```
<!--#command var=values-->
config: Determines how subsequent information will be displayed
<!--#config cmdecho=on-->
include: Includes the contents of a file.
        Can be an absolute or virtual path, so you can include dynamic information
<!--#include file="/u08/webdocs/motd.txt"-->
<!--#include virtual="/soaig/plsql/register.reg_page"-->
echo: Prints an environment variable
<!--#echo var="DATE_LOCAL"-->
fsize: Prints the size of a file -- not necessarily the file that
        includes this command
<!--#fsize file="pic3.jpg">
flastmod: Same as fsize, but prints the last time the file was modified
<!--#flastmod file="index.html">
exec: Executes a script. This could be a CGI script or a shell script.
        Administrators can disable this tag, since it does constitute a security
        risk. Note that to see the results of a cmd execution, you must use
        config to set cmd=on.
<!--#exec cgi="/cgi-bin/counter.pl">
<!--#exec cmd="/home/nick/scripts/myscript.sh">
request: Allows you to access the ICX by calling on another cartridge. Most
        commonly used to get the results of a PL/SQL, Perl or Java request, but
        can be used to call on another machine altogether, and even allows access
        to protected URL's by allowing for a username and password to be included.
<!--#request url="http://user:pass@www.mymachine.com/java/user_info?day=Sunday-->
```

提示 可以使用 LiveHTML 的 SSI 能力以允许用户改变内容，在 INCLUDEed 中以文本文件的形式，不用接触接口自身。

在 OAS 上的 LiveHTML，扩展了规模，不仅包括日期或另一个文件的内容。事实上，Oracle 含有与微软的 Active Server Pages 相似的功能，然而服务器端的脚本不是基于 VB 的，而

是基于Perl。这有几个优点，Oracle提供了许多标准的“Web应用对象”允许高级功能甚至存取CORBA对象。

内嵌脚本有三种语法形式，与ASP相似：

```
<% %> : Encapsulates a script
<%= %> : Prints the value of an expression
<SCRIPT RUNAT=SERVER> </SCRIPT> : Also encapsulates a script,
but if "RUNAT=SERVER" is left off, the script text will be provided to
and run by the browser.
```

也可以选择存取该页提供的变化，例如，假设调用网页

`http://www.myserver.com/scriptpages/hellopage.asp?user=Nick&numstuff=3`

该网页的文本包括

```
Let me guess. Your name is
<% $Response->write($user) %>
, right?
<% $Response->write(" Let's see. You own ");
    for ($countem = 1; $countem >= $numstuff; $countem++) { %>
    <%= $countem %>,
    <% } ;
    $Response->write(" computers.")%>
```

输出为：

Let me guess. Your name is Nick, right? Let's see. You own 1, 2, 3, computers.

注意 也可以选择使用<SCRIPT Language="language">指明一种语言，但现在只支持Perl语言，Oracle计划将来增加对其他语言的支持。

### 36.3.7 使用Perl插件

在应用服务器出现之前，多数CGI脚本都是用C或Perl写的，我们一直在使用CWeb与C应用打交道，现在我们准备与Perl打交道。Perl脚本比书写C应用容易得多，但将它们作为CGI运行有它的缺点。首先，像任何CGI脚本一样，每次一个用户产生一个请求，都需要启动翻译器，因为Perl是一种解释性语言，这将是一个严重的资源与性能消耗。

OAS使用Perl插件解决了这个问题，Perl插件本质上是Perl翻译器的一个版本，保留在内存中等待请求，然后执行它们。依赖于操作系统，插件作为共享对象（UNIX、libperlctx.so）或动态链接库（NT、perlnt40.dll）实现。将Perl脚本升级为OAS是一个简单的过程——基本上将虚拟路径重新映射指向插件服务器而不是CGI。

创建一个Perl应用，遵循与JWeb应用相似的步骤，选择Perl替代JWeb，文件位置不需要是任何特定环境变量的一部分。

必须紧记Perl插件的一些事项：

翻译器是基于Perl 5.003的。

标准I/O与STDERR分别重新指向浏览器与WRB记录日志。

不能使用“fork”，使用“system”，但记住输出已被重新指向浏览器。

### 36.3.8 企业级Java Beans与JCORBA

企业级Java Beans（EJB）与JCORBA包括用Java书写CORBA对象。EJB规范由JavaSoft创

建，JCORBA规范是一个Oracle拥有的规范，先于EJB由Oracle书写，最后由JavaSoft完成。它们都设计用于封装对象并允许从其他的CORBA对象发送或接收信息。选择使用哪个依赖于你已经做了什么开发及想要完成什么，紧记以下事项（见表36-2）。

表36-2 选择一个对象模型

对 象	EJB	JCORBA
现存对象	扩展java.rmi.Remote	扩展org.omg.CORBA.Object
传递对象用	值	指引
客户端的书写用	只有Java	Java或其他语言

如果使用第三方BEANS，也需要使用EJB。

## 36.4 小结

Web的早期结点只由静态信息组成，甚至没有图表。从那时以来事务已改变了许多！今天，不只Web有了巨大的进步，而且Web类型的应用也在商务的所有方面迅速增长。幸运的是，技术能跟上这个步伐，你可以使用Oracle应用服务器不只将现存的PL/SQL应用在Web上使用，而且跟上Java的步伐，甚至创建了使用C插件通常不用考虑Web内容的插件。通过这些应用服务器体系结构的优势，不论以后出现什么，都可以使用Oracle应用服务器或它的后继产品建立应用。