Oracle HTTP Server *powered by Apache*

Using mod_oprocmgr with mod_jserv

Release 1.0.2.2

June 2001

Part No. A90282-01

Introduction

This document explains how to use the module mod_oprocmgr to provide process management and load balancing services to the application processes in the Oracle HTTP Server *powered by Apache*.

Terms Used in this Document

Terms used in this document to describe the module and its functions are defined below:

Process manager

The process manager (the mod_oprocmgr module) starts, stops, and detects death of processes (starting new processes to replace them), and provides load balancing services to the processes. mod_oprocmgr gets the topology management information via HTTP requests from internal servers such as JServ, and does its job based on this information.

Group

A set of processes across which request traffic is distributed.

Servlet Engine Process

A JVM instance that runs a servlet engine, such as JServ.

Oracle HTTP Server Infrastructure

The Oracle HTTP Server infrastructure provides high availability features, process management, and performance metrics for Oracle9i Application

Server. mod_oprocmgr, DMS and Resource Monitor are components of the infrastructure.

How mod_oprocmgr Works with mod_jserv

mod_oprocmgr provides new infrastructure capabilities, such as automatic starting of processes, death detection and restart, and load balancing. These capabilities are enabled by a new mode, auto, for the ApJServManual directive.

Based on the configuration information provided by mod_jserv, mod_oprocmgr starts the specified number of JServ processes, managing them for the life of the servers.

Benefits of Using mod_oprocmgr With mod_jserv

mod_oprocmgr enhances the functionality and administration of JServ in several ways:

Process Management

With `ApJServManual off', only one JServ engine can be started and managed automatically. Additional servlet engines have to be manually started, monitored and stopped.

With `ApJServManual auto', any number of JServ engines can be started automatically. The process manager will continually monitor the health of these processes and kill and restart them, if necessary. You can still start JServ processes manually, if you need to.

Configuration

Configuring multiple JServ processes with `ApJServManual on'/`ApJServManual off' is more complicated and error prone. For example, a 10 process "balance" configuration requires 32 directives and 10 jserv.properties files.

Configuring multiple JServ processes with the new `auto' mode requires much less effort. For example, a 10 process "balance" configuration requires only 3 directives.

Flexible architecture

Without any new infrastructure, mod_jserv's functionality is limited. mod_oprocmgr makes the new Oracle9i Application Server HTTP infrastructure available to it. Therefore, its functionality can be extended in the future to accommodate dynamic configuration (by dynamically adding JServ process groups and mount points). The biggest advantage of this architecture is in a

site with multiple load-balanced HTTP servers where a new node can be dynamically added without affecting any of the other nodes.

Configuring mod_jserv for Process Management

If you are already familiar with the configuration directives for mod_jserv, the configuration process is straightforward. For detailed information on the configuration directives for mod_jserv, see the documentation at http://java.apache.org/jserv/.

Changes to httpd.conf

To use mod_oprocmgr, ensure that the directives below are included in $ORACLE_HOME/Apache/Apache/conf/httpd.conf:

```
<IfModule mod_oprocmgr.c>
  ProcNode my-sun.us.oracle.com 7777
  <IfDefine SSL>
    ProcNode my-sun.us.oracle.com 80
  </IfDefine>
  <Location /oprocmgr-service>
    SetHandler oprocmgr-service
  </Location>
 </IfModule>
```

In addition, you must specify at least one non-SSL port. For a secure website (that is, one that only accepts SSL connections), you must provide an extra non- SSL port. To do this, add the directives shown below, substituting port and address values:

```
Listen <port>
<VirtualHost _default_:port>
   SSLEngine Off
  <Location />
    order deny, allow
    deny from all
    allow from <IP address 1 of local node>
    allow from <IP address 2 of local node>
    allow from <IP address 3 of local node>
  </Location>
</VirtualHost>
```

Changes to jserv.properties

In the jserv.properties file, found in $ORACLE_HOME/Apache/Jserv/etc/, you specify the ports to which JServ will bind, as shown in the example below. If none are specified, the JServ processes will choose their ports.

```
port=8007
```

You can specify multiple ports, and separate the values with commas as shown in the example below. Note that a range of ports (9000-9010) is a valid value.

```
port=8007,9000-9010,8010
```

Changes to jserv.conf

To use mod_oprocmgr with mod_jserv, you must change the directives as indicated below in the JServ configuration file jserv.conf, in:

$ORACLE_HOME/Apache/Jserv/etc/

ApJServManual This directive accepts a new mode, auto, which invokes the new infrastructure functionality (in which mod_oprocmgr manages processes). The syntax is:

```
ApJServManual auto
```

You can set the mode to on or off to use the standard JServ functionality.

ApJServGroup This new directive defines groups for the process manager to manage for mod_jserv. If you have worked with mod_jserv, you will note that this directive replaces the ApJServBalance, ApJServHost, ApJServRoute and ApJServShmFile directives.

All JServ processes to be managed must belong to a group, and each group has its own ApJServGroup directive. If you only have one JServ process, you must define a group with just that process in it. The processes in a group are identical except for their listening ports, so requests directed to the group are distributed evenly among the processes.

The ApJServGroup directive takes four arguments: groupname, number of processes, node weight, and properties file. In the example below, the groupname is mygroup, the number of processes is 1, the node weight is 1, and the full path of the properties file used to start the JServ processes is /private2/up_1022/Apache/JServ/etc/jserv.properties.

```
ApJServGroup mygroup 1 1 /private2/up_1022/Apache/Jserv/etc/jserv.properties
```

The node weight argument is relevant in a multi-node configuration (planned in a future release). Since the same group can be started on multiple nodes, the node weight is used for weighted load balancing among nodes for requests to that group.

ApJServGroupMount This directive defines a mount point and maps it to a process group and zone. In the example below, the mount point is /servlets,

the group is mygroup, and the zone is root. Note that the balance protocol is in use for routing, as in the standard JServ configuration.

```
ApJServGroupMount /servlets balance://mygroup/root
```

Place this directive after the ApJServGroup directive in the configuration file.

ApJServGroupSecretKey This directive specifies the secret key that JServ needs to authenticate clients. It can be disabled, as shown below:

```
ApJServGroupSecretKey disabled
```

When activated, the directive takes one or two arguments. In the example below, with group and filename arguments, the filename mysecretkey applies to the group mygroup.

```
ApJServGroupSecretKey mygroup /usr/local/apache/jserv/mysecretkey
```

You can supply only the filename argument, as shown below. No group is named, so the secret key filename applies to all groups.

```
ApJServGroupSecretKey /usr/local/apache/jserv/mysecretkey
```

You cannot combine directives using the one-argument syntax with directives using the two-argument syntax. If you use the two-argument syntax, the default for groups without a group-specific secret key is 'disabled'.

Place this directive after the ApJServGroup directive in the configuration file.

Warning: The secret in the secret key file specified in ApJServSecretKey must be the same as that specified by the security.secretKey directive in the jserv.properties file. If the secrets are not the same, the death detection mechanism assumes that all the servlet engine processes are dead, eliminates them, and starts new processes to replace them (repeating the cycle endlessly).
