

优化 WebLogic

一、为 WebLogic 启动设置 Java 参数

垃圾收集(GC)是指 JVM 释放 Java 堆中不再使用的对象所占用的内存的过程,而 Java 堆(Heap)是指 Java 应用程序对象生存的空间。堆大小决定了 GC 的频度和时间。堆越大,GC 频度低,速度慢。堆越小,GC 频度高,速度快。所以 GC 和堆大小是一组矛盾。为了获取理想的 Heap 堆大小,需要使用-verbosegc 参数(Sun jdk: -Xloggc:<file>)以打开详细的 GC 输出。分析 GC 的频度和时间,结合应用最大负载所需内存情况,得出堆的大小。

通常情况下,我们建议使用可用内存(除操作系统和其他应用程序占用之外的内存)70-80%,为避免堆大小调整引起的开销,设置内存堆的最小值等于最大值即:-Xms=-Xmx。而为了防止内存溢出,建议在生产环境堆大小至少为 256M(Platform 至少 512M),实际环境中 512M~1G 左右性能最佳,2G 以上是不可取的,在调整内存时可能需要调整核心参数进程的允许最大内存数。对于 sun 和 hp 的 jvm,永久域太小(默认 4M)也可能造成内存溢出,应增加参-XX:MaxPermSize=128m。建议设置临时域-Xmn 的大小为-Xmx 的 1/4~1/3, SurvivorRatio 为 8

堆栈内存优化, 修改配置文件:

```
WL_HOME=C:\bea\weblogic81 "%WL_HOME%\common\bin\commEnv.cmd"
:bea #如果采用的上 bea 的 JDK
# JVM Heap (堆内存) 最小尺寸为96M, 最大尺寸为256M
set MEM_ARGS=-Xms96m -Xmx256m
:sun #如果采用的是 sun 的 JDK
# JVM Heap (堆内存) 最小尺寸为32M, 最大尺寸为200M
#公共变量对象的内存限制: PermSize: 最小尺寸, MaxPermSize : 最大允许分配尺寸
set MEM_ARGS=-Xms32m -Xmx200m -XX:MaxPermSize=128m
```

监视堆栈使用情况:

下载JRockit JDK,该JDK已经自带了JRockit Mission Control工具,目前好像还没有单独下载JRockit Mission Control的地方,于JRockit JDK进行了绑定下载;

在 C:\bea\jrockit81sp5_142_08\console 目录里面运行:

```
C:\bea\jrockit81sp5_142_08\bin\java -Xmanagement -jar ManagementConsole.jar
```

如何监控weblogic呢?

修改weblogic启动脚本startWebLogic.cmd, 在里面加入-Xmanagement启动参数:

```
%JAVA_HOME%\bin\java -Xmanagement %JAVA_VM% %MEM_ARGS% %JAVA_OPTIONS%
-Dweblogic.Name=%
SERVER_NAME% -Dweblogic.ProductionModeEnabled=%PRODUCTION_MODE%
-Djava.security.policy="%
WL_HOME%\server\lib\weblogic.policy" weblogic.Server
```

二、设置与性能有关的配置参数

在一个 WebLogic 域中，配置文件（config.xml）位于与管理服务器通信的机器里，提供 WebLogic MBean 的长期存储。管理服务器作为连接的中心点，为服务实例与系统管理工具提供服务。域也可以包括其他的 WebLogic 实例，称之为从服务，主要为应用程序提供服务。当启动管理服务器时，首先读域配置文件，然后跳过建立在配置文件中管理 MBean 默认的属性值，每一次用系统管理工具（不管是命令行界面还是管理控制台）改变一个属性值，它都会被存到相应的管理 MBean，并且写进配置文件。

1. 下表列出了影响服务器性能的参数。

名称	参数	位置	作用
高速缓存	max-beans-in-cache	weblogic-ejb-jar.xml	实体 bean 缓存空间的大小，如果缓存的空间太小，有些 bean 就被滞留在数据库中，下次调用时就必须重新从数据库装载
	"%JAVA_HOME%\bin\java" -hotspot -Xms512m -Xmx512m -classpath %CLASSPATH% -	startWLS.cmd /startWLS.sh	为得到高性能的吞吐量，把 Java 堆的最小值与最大值设为相等。

2.console控制台中的参数

名称	类型	位置	值
NativeIOEnabled	server	mydomain->Servers->myserver->Configuration->Tuning->“Enable Native IO”	TRUE，表示该 Server 使用本地 I/O
SocketReaders		server->configuration->tuning	设置在执行线程中专用做 Socket Readers 的百分比
Maximum Open Sockets			最大打开 Socket 数
Stuck Thread MaxTime			堵塞线程时间，超过这个时间没有返回的执行线程，系统将认为是堵塞线程 如果 weblogic 认为某个队列中的所有线程全部堵塞的话，weblogic 将会增加执行线程的数

			<p>量。</p> <p>注意：执行线程的数量一旦增加，目前 weblogic 不会去减少他，如果增加了一些线程以后再次出现 overflow 的警告，weblogic 会继续增加执行线程的数量，一直到达到上限为止。</p>
Stuck Thread Timer Interval			系统检查堵塞线程的时间间隔
Low Memory GC Threshold			当可用内存小于该百分比时，垃圾回收启动
Low Memory Granularity Level			当两次检测的可用内存变化超过该百分比时，垃圾回收启动
Low Memory Sample Size			在一次检测中的取样次数
Low Memory Time Interval			检测间隔时间
Accept Backlog			<p>等待队列中最多可以有多少 TCP 连接等待处理，如果在许多客户端连接被拒绝，而在服务器端没有错误显示，说明该值设得过低。</p> <p>如果连接时收到 connection refused 消息，说明应提高该值，每次增加 25%</p>
ThreadCount	ExecuteQueue	<p>console: mydomain->Servers->myserver ->Monitoring->Monitor all Active Queues...</p> <p>->Configuration->weblogic.kernel.Default-></p>	<p>服务器初始创建的执行线程的数量，设置原则：增大机器的最大并发线程数使处理器利用率达到最大。对于服务器端操作比较多</p>

			的线程，应该减少线程计数；对于客户端操作比较多的，应该增加线程计数。并发线程数理论上等于“本地主机 CPU 个数+Stuck 线程数”，够用即可，过大会降低系统性能
QueueLength			在等待队列里的请求数，理想状态下是 0
QueueLengthThresholdPercent			一个百分数，当 request 的数量达到队列长度的这个比例的时候，weblogic 会发出 overflow 的标志信息
ThreadsIncrease			如果 weblogic 发出 overflow 的标志信息，weblogic 会尝试增加这个数量的执行线程，以解决处理矛盾
ThreadsMaximum			最大执行线程数
ThreadsMinimum			最小执行线程数
ThreadPriority			线程优先级
Initial Capacity	JDBC	mydomain-> JDBC Connection Pools->Configuration->Connections	初始数据库物理连接数
MaxCapacity			最大数据库物理连接数
Capacity Increment			每次数据库物理连接增加数
Statement Cache Type			prepared statements 缓存的策略，LRU 算法在有新的语句到来时，将最不经常被用得语句调整出缓存。FIXED 算法为先进先出的算法

TestConnectionsOnReserve		TestConnectionsOnReserve 设置为 false (缺省设置)。如果此参数设置为真 (true), 则在连接被分配给调用者之前, 都要经过测试, 这会额外要求与数据库的反复连接
Statement Cache Size		宏语句设定的静态缓存, 大小由 JDBC 连接池配置时指定, 调整这个数值的大小, 有利于提高系统的效率
Login Delay		创建数据库物理连接时的延时时间

三、调整开发模式与产品模式默认值

你可以指定域为开发环境或为产品环境。WebLogic 会根据你指定的环境类型使用不同的默认值提供不同的服务。

下表列出了两种模式下的默认值

优化参数	开发模式	产品模式
mydomain>Servers>myserver>Execute Queue>weblogic.kernel.Default 中 Thread Count	15 threads	25 threads
mydomain> JDBC Connection Pools> MyJDBC Connection Pool-> Connections 中 Maximum Capacity	15 connections	25 connections

3. 1 更改运行时模式

在创建了一个域后, 按下列步骤可以更改域里所有服务的的运行时模式:

1. 为更改运行在一个 WebLogic 主机上的所有域的运行时模式, 用文本编辑器打开 WL_HOME\common\bin\commEnv.cmd(Windows) 或者 WL_HOME\common\bin\commEnv.sh (UNIX), WL_HOME 是安装 WebLogic 的路径。

为指定的域更改运行时模式, 就用文本编辑器打开 domain-name\StartWebLogic.cmd (Windows) or domain-name\StartWebLogic.sh (UNIX), domain-name 为创建的域的目录。

2. 在这个脚本中, 更改 PRODUCTION_MODE 的值, 如果你要服务器运行在产品模式, 指

定其值为 TRUE。或者在 mydomain-> General 里面把 Production Mode 选中
更改为产品模式时，并且把 boot.properties(存放着 username 和 password)文件删掉，那么启动时就需要输入用户名和密码。

3. 重启所有的服务器。

3. 2 两种模式的不同

下表列出了开发模式与产品模式几种关键项的区别：

功用名称	开发模式	产品模式
SSL	你可以使用 WebLogic 安全服务提供的验证数字证书。有这些证书，你开发的应用程序会在 SSL 保护的环境下运行。	如果你使用验证数字证书，会收到警告信息。
部署应用程序	WEBLOGIC 实例会自动部署和更新位于 domain_name/applications 目录下的应用程序（domain_name 为域的名称）。	不能使用自动部署功能，必须使用 WebLogic 控制台或者 WebLogiceblogic Deployer 工具。
Log File Rotation	启动服务器后，服务器自动重命名本地日志文件为 server-name.log.n, 为了滞留的 session，只要日志文件的达到 500kb，日志文件就会滚转一次。	当日志文件达到 500kb，就会滚转。
Execute Queues	默认的执行线程为 15。	默认的执行线程为 25。
JDBC Connection Pool Capacity	默认容量为 15。	默认容量为 25。

四、使用 WebLogic “自有的 IO” 性能包

当你使用自有的性能包，测试基准就表明了主要性能的提高。性能包采用最优化的平台及多线程的 Socket 去提高服务器的性能。例如，本地 Socket 读的多线程有自己的执行队列而不需要借用默认的执行队列线程，这样可以让默认执行线程很从容去处理应用程序。

不过，如果你一定要用纯 Java socket 读在主机上运行，你仍然可以通过配置每个服务器实例和客户机中适当的 socket 读的线程数量，来提高 socket 通信的性能。

设置性能包的操作方法：

默认情况下，装载在 config.xml 中的是自有的性能包。为了验证这个设置，在配置文件中检

查 NativeIOEnabled 属性是否设为 “true” (NativeIOEnabled=true)。

你也可以通过管理控制台来验证，步骤如下：

- 1, 启动管理服务器。
- 2, 访问管理控制台。
- 3, 展开左边面板的 Servers 节点，显示域服务。
- 4, 点击你要配置的服务实例。
- 5, 选择 Configuration -> Tuning tab。
- 6, 如果 Enable Native IO 复选框没有被选择，选中即可。
- 7, 点击 Apply。
- 8, 重启服务器。

五、优化默认执行队列线程

默认情况下，一个新的 WebLogic 实例配置了一个开发模式执行队列，weblogic.kernel.default，它包含 15 个线程。另外，WebLogic 提供了 2 个预配置队列：

n weblogic.admin.HTTP——只在管理服务器上才有，这个队列供与管理控制台的通信用，你不能再配置它。

n weblogic.admin.RMI——管理服务器和被管理服务器上都有这个队列，它是供管理的交通之用，也不能再配置它。

如果你不配置额外的执行队列，并且指定应用给这些队列，web 应用程序和 RMI 对象就使用默认的队列 weblogic.kernel.default。

注意：如果自带的执行包没有在你的平台上使用，你可能需要调整默认的执行队列线程数和担任 socket 读的线程的百分比，去实现最佳性能。

5.1 你应该更改默认的线程数吗？

增加更多的线程到默认的执行队列并不意味着你能处理更多的工作。即使增加更多的线程，仍然被处理器的能力限制。因为线程消耗内存，所以增加线程数属性的值不必要的降低了性能。一个高的执行线程数导致更多的内存被占用并且增加了上下文转换程序，它也会降低性能。

线程数属性的值与应用程序处理的工作的类型关系密切。例如，如果你的客户应用程序比较小，通过远程调用处理的工作较多，这样，客户端会花费更多的时间连接，因此，与能完成大量客户端任务的客户应用程序相比，会需要更多的线程数。

如果你的工作不需要使用超过 15 个线程（开发模式默认）或者 25 个线程（产品模式默认），就不要改变这个属性的值。通常，如果你的应用程序访问数据库花很长时间才返回结果，与访问数据库很短时间就返回的应用程序比较，你会需要更多的执行线程。从后者来看，用少点的线程数可能提高性能。

5. 2 需要修改默认线程数的情形

为了给执行队列决定一个理想的线程数，当队列中所有应用程序都运行在最大负荷的情况下，监视队列的吞吐量。增加线程数，重复负载测试，直到达到最佳的吞吐量。（在某些情况下，增加线程数将产生足够多的上下文转换程序，使得队列中的吞吐量开始减少。）

注意：WebLogic 管理控制台显示的是所有服务器执行队列累积的吞吐量。为了得到这个值，后面将会介绍。

下表列出了在 WebLogic 域中调整的线程及与 CPU 数量相关的情形，这些情况也假定 WebLogic 运行在最大负荷下，并且使用默认的执行队列满足所有的线程的请求。如果你配置了额外的执行队列并指派了应用程序到具体的队列，就需要依据一个个连接池得到结果。

如果... 结果 应该：

线程数 < CPU 的数量 线程数太少，如果：

CPU 正等着工作，但有工作被完成。

CPU 利用率不能达到 100%。 增加线程数。

线程数 = CPU 的数量 理论上理想，但是 CPU 仍然低利用。 增加线程数。

线程数（适当的） > CPU 的数量 实际中理想，有个适当的上下文转换程序数量和高的 CPU 利用率。 调整适当的线程数并且比较性能结果。

线程数（较大的） > CPU 的数量 过多的上下文转换程序，能导致重大的性能降级。

当你降低线程数时，性能可以增强。 减少线程数，使它等于 CPU 的数量，然后仅仅增加已经得出的“堵塞”线程的数量。

例如，如果你有 4 个处理器，它们都同时运行，并出现堵塞线程，于是，你想要的执行线程就是 4 + 堵塞线程的数

5. 3 修改默认线程数的步骤

用管理控制台修改默认执行队列线程数如下：

1. 如果管理服务器没有运行，先启动。
2. 访问管理控制台。
3. 展开左边面板的 Servers 节点，显示域服务。
4. 右击服务名称，在弹出菜单中选择 View Execute Queues ，就会在右边面板显示有执行队列的表用来修改。

注意：你只能修改默认的执行队列或者用户定义的执行队列。

5. 在 Name 列，直接点击默认执行队列名称，显示配置标签用来修改执行队列数。
6. 填下适当的线程数。
7. 点击 Apply，保存刚才的修改。
8. 重启服务器，使新的执行队列设置生效。

5. 4 指派应用程序到执行队列

虽然可以配置默认的执行队列，为所有的 WebLogic 应用程序提供最佳的线程数，但是为关键的应用程序配置多个执行队列可以提供更多的管理控制。通过使用多执行队列，你可以保

证应用程序有权占用固定的线程数，而不管 WebLogic 服务器有多大的负荷。

5. 5 创建执行队列

一个执行队列代表执行线程的命名集，线程指向一个或多个 Servlet、JSP、EJB、RMI 对象。执行队列在 config.xml 文件中描述，作为服务器元素的一部分。如，在 config.xml 文件中描述一个有 4 个线程的队列，命名为 CriticalAppQueue，如下：

```
...
<Server
Name="examplesServer"
ListenPort="7001"
NativeIOEnabled="true"/>
<ExecuteQueue Name="default"
ThreadCount="15"/>
<ExecuteQueue Name="CriticalAppQueue"
ThreadCount="4"/>
...
</Server>
```

另一种创建队列的方法是通过管理控制台，配置步骤如下：

1. 启动管理服务器，访问控制台。
2. 展开左边面板中 Servers 节点，显示域中要配置的服务。
3. 右击你要增加队列的服务实例，从弹出菜单中选择 View Execute Queues。
4. 在队列配置标签中，点击配置新执行队列链接。
5. 在队列配置标签中，更改下列属性或接受系统的默认值：
 - 线程名称 (Name)：你可以输入线程名称，如 CriticalAppQueue。
 - 队列长度 (Queue Length)：通常保留默认值 65536，队列长度表明了同时发来请求的最大数，65536 个请求是个很大的数，即使达到这个最大数，也是很少见的。
 - 如果达到最大队列长度，WebLogic 会自动成倍增长队列大小，以处理额外的工作。注意：超过 65536 个请求预示队列中的线程有问题，不仅仅只是队列本身的长度问题，实践表明在队列中有堵塞线程或线程数不足的情况存在。
 - 队列长限制百分比 (Queue Length Threshold Percent)：达到队列长度百分比 (1-99) 时，就构成了溢出条件的产生。实际队列大小在限制的百分比之下时才被认为是正常的；在限制百分比之上就会产生溢出。当出现溢出，WebLogic 日志就会产生一个错误消息，并且按线程数增量 (Threads Increase) 属性的值增加线程数，以帮助减少负载量。
 - 默认的队列长限制百分比为 90%。一般情况下，应保留 90% 或其左右，以应对一些潜在的情况，使得有额外的线程可以去处理一些请求中的异常。记住，队列长度限制百分比不是一定作为自动优化参数——因为正常运作情况下，这个限度从不会被触发。
 - 线程数 (Thread Count)：指派到这个队列的线程数。如果你不需要使用超过 15 个线程 (默认)，就不必更改这个属性值。
 - 线程数增量 (Threads Increase)：是指 WebLogic 探测到有溢出时，增加到执行队列的线程数。当你指定为 0 (默认)，出现溢出时，WebLogic 会把运行良好状态改为“警告”，而且也不会指派额外的线程去减少负荷量。

- 注意：如果 WebLogic 实例的线程数响应了溢出，那么这些额外的线程就会滞留在执行队列，直到服务器重启。监视错误日志，以判断溢出产生的原因，以便根据需要重配置线程数，防止以后类似情况产生。不要同时使用线程数增量和队列长限制百分比作为自动优化的手段。如此做通常结果会产生比正常需要还多的线程被指派到执行队列，这样上下文转化程序的增多会使服务器遭受很差的性能。
 - 最大线程数：是指执行队列中能运行的，这个值保护 WebLogic 为了响应频繁溢出，创建过多的线程数。默认情况下，最大线程数为 400。
 - 线程优先级：线程优先级与此队列相关。默认值为 5。
6. 点击 Create，创建队列。
 7. 重启服务器。

5. 6 指派 Servlet 和 JSP 到执行队列

你可以把 servlet 或 JSP 分配到指定的配置执行队列，只需在初始参数中标识执行队列的名称。初始参数出现在 Servlet 或 JSP 的部署描述文件 web.xml 中的 init-param 元素里。为了分配一个队列，可以把队列名作为 wl-dispatch-policy 参数的值。如：

```
<servlet>
```

```
    <servlet-name>MainServlet</servlet-name>
```

```
    <jsp-file>/myapplication/critical.jsp</jsp-file>
```

```
    <init-param>
```

```
        <param-name>wl-dispatch-policy</param-name>
```

```
        <param-value>CriticalAppQueue</param-value>
```

```
    </init-param>
```

```
</servlet>
```

5. 7 指派 EJB 和 RMI 对象到执行队列

为了把 EJB 分配到指定的队列，可以使用 `weblogic-ejb-jar.xml` 文件中 `dispatch-policy` 元素。然而你也可以通过使用 `appc` 编译器 `-dispatchPolicy` 选项来设置派遣策略，BEA 强烈推荐使使用部署描述元素。因为用这种方式，如果 EJB 重编译，在部署用例期间，这个设置不会被丢失。

为了把 RMI 对象分配到指定的队列，可以使用 `rmic` 编译器 `-dispatchPolicy` 选项，如：
`java weblogic.rmic -dispatchPolicy CriticalAppQueue ...`

5. 8 分配执行队列担任 Socket 读

为了获得更好的 socket 性能，BEA 推荐你使用自有的 socket 读执行工具，它更优于纯 Java 执行工具。然而，如果你一定要在主机上用纯 Java 的 socket 读，你仍然可以通过配置恰当的线程数以提高 socket 通信性能，为每个服务器实例和客户机器担负 socket 读线程的任务。

Socket 读占线程池百分比 (`ThreadPoolPercentSocketReader`) 属性可以设置用来从 socket 读消息的执行线程的最大百分比。这个属性的最优值是根据应用程序的需要指定的。默认值是 33，有效范围在 1-99 之间。

分配线程担任 socket 读增加了服务器处理速度和接受客户请求的能力。有必要平衡线程数，使其专注于从 socket 读消息，也有必要平衡那些在服务器处理实际任务的线程。

5. 9 为服务器实例设置 socket 读的线程数的操作

1. 启动管理服务器，访问域控制台。
2. 展开左边面板 `Servers` 节点，显示域服务配置。
3. 点击你要配置的服务名称。
4. 选择配置 (Configuration) --> 调整 (Tuning) 标签。
5. 在 `Socket Reader` 中编辑 Java 读线程的百分比。Java socket 读线程数是根据所有的线程数的百分比计算得到的。
6. 应用 (Apply) 这个调整。

5. 10 在客户机设置 Socket 读线程数

在客户机上，你可以配置运行在 JVM (Java 虚拟机) 上的 socket 读线程数。指定 Socket 读，需要通过用 `java` 命令行定义下列参数：

`-Dweblogic.ThreadPoolSize=value`

`-Dweblogic.ThreadPoolPercentSocketReaders=value`

5. 11 优化溢出情况时的执行队列

你可以配置 **WEBLOGIC** 监测并且随时应对潜在的溢出，不管其发生在默认的执行队列还是用户定义的队列。一旦当前队列大小快达到用户定义的百分比，**WebLogic** 认为队列中有一个可能的溢出产生。

当这个限度到达时，服务器改变它的良好状态为“警告”，随即分配额外的线程去处理超负荷的工作，从而还原它的大小。

为了自动监测和应对溢出，你可以配置以下项：

1. 队列长限制百分比，这个值是队列大小的百分比。
2. 当溢出发生时，增加到队列的线程数。这些额外的线程以还原队列到正常的运行的大小。
3. 线程的最大数，在特殊情况下，线程最大数用来保护服务器在响应过载情况下过度分配线程数。

5. 12 优化执行队列的监测行为

当一个线程在队列中变成堵塞状态时，**WebLogic** 会自动监测到。因为堵塞线程不能完成它当前的工作或接受新的工作，服务器每次诊断一个堵塞线程，就记入一个消息到日志中。如果一个队列中所有的线程变成堵塞，服务器改变良好状态成“警告”或者“危机”，依赖于下列情况：

n 如果默认队列中所有的线程变成堵塞，服务器状态变成“危机”。（你可以设立节点管理器（Node Manager）应用去自动关闭及重启服务器。）

n 如果在 `weblogic.admin.HTTP`、`weblogic.admin.RMI` 或用户定义的队列中所有线程变成堵塞，服务器状态变成“警告”。

WebLogic 诊断到一个堵塞线程，如果它是在指定的时间内连续不断的工作（没有空闲）。你可以调整服务器线程监测行为，它是通过改变堵塞线程被诊断前的时间长度和服务器核查堵塞线程的频率。

注意：尽管你能改变标准 **WebLogic** 去决定一个线程是否堵塞，但，你不能改变默认行为，就是出现堵塞时把服务器设置成“警告”或“危机”的行为。

配置 **WebLogic** 堵塞线程监测行为的步骤：

1. 启动 **WebLogic**，访问管理控制台。
2. 点击你想为改善堵塞线监测而修改的服务器实例的名称。
3. 选择配置（Configuration）——>调整（Tuning）标签。
4. 修改下列参数：

n 堵塞线程最大时间（Stuck Thread Max Time）：输入秒数，线程一定是不断的运行，服务器才会诊断这个线程作为堵塞。默认情况下，**WebLogic** 认为线程连续不断运行 600 秒后置为堵塞。

n 堵塞线程时间间隔（Stuck Thread Timer Interval）：输入秒数，这个时间是 **WebLogic** 周期性的扫描线程以察觉它们是否连续不断运行了某一线程的时间达到通过堵塞线程最大时间属性指定的时间长度。默认时间间隔为 600 秒。

5. 应用（Apply）设置。
6. 重启服务器。

5. 13 设置 HTTP 提交的超时时间

在 weblgoic 的 console 中：MyDomain—>Servers->MyServer->Protocols->HTTP 中有一个关于 Post Timeout 的配置，用来设置用 HTTP 提交的时间，但这个参数一般使用默认值即可

六、优化连接缓存

Config.xml 文件中的元素接受缓存数（AcceptBacklog）属性是用来设定请求 WebLogic 实例的连接数，在拒绝额外的请求之前，能接受设定的缓存数。AcceptBacklog 属性指定有多少 TCP 连接缓存在等待队列，这个固定的队列存放了 TCP 堆栈已经收到但应用程序还没有收到的连接请求。默认值是 50，最大值由操作系统决定。

在控制台调整接受缓存数的步骤：

1. 启动 WebLogic，访问控制台。
2. 展开左边面板 Servers 节点。
3. 点击你要配置的服务器实例的名称。
4. 选择配置（Configuration）——>调整（Tuning）标签。
5. 根据需要修改默认接受缓存数（Accept Backlog）：
 - n 在运行期间，如果许多客户端连接得不到响应或被拒绝，并且服务器端也没有错误消息，说明接受缓存的值可能太小。
 - n 在你访问 WebLogic 时，如果收到“拒绝连接（connection refused）”的提示，则应该增加接受缓存的默认值的 25%。继续增加其值的 25%，直到停止出现这样的提示。
6. 点击应用（Apply），保存设置。

七、如何提高 JDBC 连接池的性能

创建一个带 DBMS 的 JDBC 连接是非常慢的。如果应用程序需要数据库不断的连接和断开，这种创建方式会造成一个重大的性能问题。WebLogic 连接池提供了一种高效的解决方案来解决这个问题。

当启动 WebLogic，就打开连接池，以便于所有客户连接。当一个客户关闭一个连接，这个连接就返回到连接池，供其他的客户使用。连接本身不会关闭。如此就用极少的代价实现了连接和断开连接池。

在连接池里应该创建多少连接呢？连接池会根据配置参数中的最大数与最小数之间增加或减少连接。最好的性能应该是连接数与当前客户会话（Session）数相同。

7. 1 调整 JDBC 连接池的初始容量

在配置连接池时，JDBCConnectionPool 元素中的 InitialCapacity 属性能设定连接数，创建物理的数据库连接。如果服务器不能创建这个连接数，连接池的创建就会失败。

在开发期间，为了使服务器启动更快，可以很方便的设置 InitialCapacity 属性的值小一点。

在产品系统中，就应该把 `InitialCapacity` 的值设为与 `MaxCapacity` 值相同，默认产品模式的值为 25。这样，在服务器启动时，所有的连接就会被创建。如果你调整了 `MaxCapacity` 值后，一定要确信 `InitialCapacity` 值设置与 `MaxCapacity` 值相同。

如果 `InitialCapacity` 比 `MaxCapacity` 值少，当负荷增加时，服务器需要创建额外的数据库连接。当服务器处于低负荷时，所有的资源应该是尽快的完成请求，而不是创建新的数据库连接。

7. 2 调整 JDBC 连接池的最大容量

`JDBCConnectonPool` 元素中的 `MaxCapacity` 属性设置连接池包含的最大的物理数据库连接数。不同的 JDBC 驱动程序和数据库服务器可能限制物理连接数。

默认的最大容量数与默认的线程数相等：开发模式为 15，产品模式为 25。不过，在产品模式下，建议连接数与当前的客户会话（Session）数相等。在服务器端，连接池的容量与执行线程数是无关的，正在进行的用户会话比执行线程更多。

7. 3 保留 JDBC 连接和回收不活动的连接

- 通过 Services→JDBC→Connection Pools→MyConnection(你所建立的连接池名)→Configuration→Connections(Advanced Options) 的 Inactive Connection Timeout 这个参数来设置的，默认的为 0，表示连接时间无限长。你可以设一个时间值，连接超过这个时间值，它会把连接强制放回连接池。
- 通过 Services→JDBC→Connection Pools→MyConnection(你所建立的连接池名)→Configuration→Connections(Advanced Options) 里的 Connection Reserve Timeout，来设置连接保持时间，默认是 10s

八、设置 Java 编译器

编译 JSP Servlet 的标准 Java 编译器是 `javac`。你可以把 java 编译器设置为 `si` 或 `jikes` 代替 `javac`，这样能极大的提高性能。下面讨论设置步骤及其要考虑的事项。

8. 1 通过控制台改变编译器

1. 启动服务器，访问控制台。
2. 展开左边面板 Servers 节点。
3. 点击要配置的服务器实例的名称。
4. 选择配置（Configuration）→常规（General），在 Java Compiler 编辑框输入编译器的完全路径。如：`c:\visualcafe31\bin\sj.exe`
5. 点击高级选项（Advanced Option）→Show，显示其他的属性。
6. 用添加（Append）把完全路径通过 Classpath 框输入到 JRE rt.jar 库。如：`BEA_HOME\jdk141_02\jre\lib\rt.jar`
7. 点击应用。
8. 重启服务器。

8. 2 在 Weblogic.xml 文件中设置编译器

- n 使用 compileCommand 参数指定 Java 编译器。
- n 使用 procompile 参数配置 WebLogic，在启动 WebLogic 时预编译 JSP。

```
<jsp-descriptor>
<jsp-param>
  <param-name>compileCommand</param-name>
  <param-value>javac</param-value>
</jsp-param>
</jsp-descriptor>
```

8. 3 编译 EJB 容器类

使用 Weblogic.appc 的功能去编译 EJB2.0 和 1.1 容器类。如果编译 Jar 文件部署 EJB 容器，你必须使用 weblogic.appc 生成容器类。默认情况下，EJB 使用 javac 编译器。为了得到跟好的性能，使用 -compiler 标志指定不同的编译器（如 Symantec 公司的 sj）

8. 4 在 UNIX 环境下编译

在 UNIX 机器上编译 JSP 文件，如果收到下列错误消息：

failed: java.io.IOException:Not enough space

试试下列一些或所有的解决方法：

- n 如果你只有 256MB 的内存，增加更大的内存。
- n 提高文件描述文件的限制，如：
set rlim_fd_max=4096
set rlim_fd_cur=1024
- n 启动 JVM 时，用 -native 标志来使用自有的线程。

九、使用 WebLogic 集群提高性能

WebLogic 集群是指一组 WebLogic 实例在一起提供具有防过载和自有复制的功能，以用一个域为所有客户支持可伸缩的高可用性运行。集群对于客户是一个单一的服务器，但实际上是一组服务器来提高可靠性和可伸缩性。

9. 1 可伸缩性和高的可用性--集群

可伸缩性是系统增加一个或更多部件作为系统资源的能力。很典型的是，这些部件使并发用户得到支持，使并发事务能在特定的时间单位能被处理。

假定应用程序设计良好，它完全可以简单的增加更多的资源来提高性能。为了增加 WebLogic 处理的负荷量，只需增加一个 WebLogic 实例到你的集群——不需改变应用程序。集群提高两个关键的好处：可伸缩性和可用性，这是单一服务器无法比拟的。

WebLogic 集群给 J2EE 带来了可伸缩性和高的可用性，而且对于应用程序的开发者是透明

的。可伸缩性扩展了中间层的能力，超过了单一的 WebLogic 服务器或单一的计算机能处理的。集群成员唯一的限制是所有 WebLogic 必须要用 IP 多点传送通信。新的 WebLogic 能动态的增加到集群，以增加处理能力。

WebLogic 集群保证高的可用性是通过多个服务器的冗余，减少客户的请求失败。集群中多个服务器能提供同一服务。如果一个服务器停止运行，另一个能接替运行。这种功能为客户增加了可用性。

警告：如果你要解决应用程序和环境的瓶颈问题，增加额外的服务器到集群，应该提供线性的可伸缩性。定基准和初始配置测试运行时，**在移到集群环境之前，应把应用隔离在单独的服务器上测试。**

9. 2 在多 CPU 机器上运行多服务器实例，应考虑的性能问题

多处理器的机器上，必须考虑群集 WebLogic 实例数应与 CPU 的数量成比例。因为 WebLogic 没有内置限制的服务器实例数位于集群里，规模大的、多处理器服务器，如 Sun 公司的 Sun Enterprise 10000，有着当作非常大的集群或多集群主机的潜能。

在决定最佳的服务器与 CPU 比例前，彻底测试你的应用程序并确定如下：

- 网络要求 如果你发现 Web 应用程序是主要受网络 I/O 限制，在增加 CPU 数前，考虑测试网络的吞吐量。如果实际是网络 I/O 限制，安装一个更快的网络接口卡（NIC）可以提供性能，而不是增加额外的 CPU，因为在等着读 socket 时，更多的 CPU 会处于闲置。
- 磁盘 I/O 要求 如果你发现 Web 应用程序主要受磁盘 I/O 限制。在配置额外的 CPU 前，就应该考虑增加磁盘转速或单个磁盘。
- 总之，在配置额外的 CPU 前，必须确定 Web 应用程序是受 CPU 限制，而不是受网络或磁盘 I/O 限制。

对于受 CPU 限制的应用，最初在每个 CPU 上对一个 WebLogic 实例进行性能测试。如果 CPU 利用率是一致的或者接近 100%，然后增加 CPU 比重（例如，为一个 WebLogic 实例配置两个 CPU），记住在产品模式下，应该有一些空闲的 CPU 周期存在去执行管理任务。

虽然 Web 应用程序的处理需求变化多端，但 **BEA 公司发现 WebLogic 实例与 CPU 最理想的比例是 1: 2。**

十、监视 WebLogic 域

监视 WebLogic 域的健康状况和性能的工具是管理控制台。通过控制台，你可以观察到 WebLogic 资源的状态和统计信息，如服务器，HTTP，JTA 子系统，JNDI,安全，CORBA 连接池，EJB，JDBC 以及 JMS。

举个例子，在 Server——>Monitoring——>Performance 为当前服务器实例提供了与等待和运行状态的请求有关的性能参考。它包括下列信息：

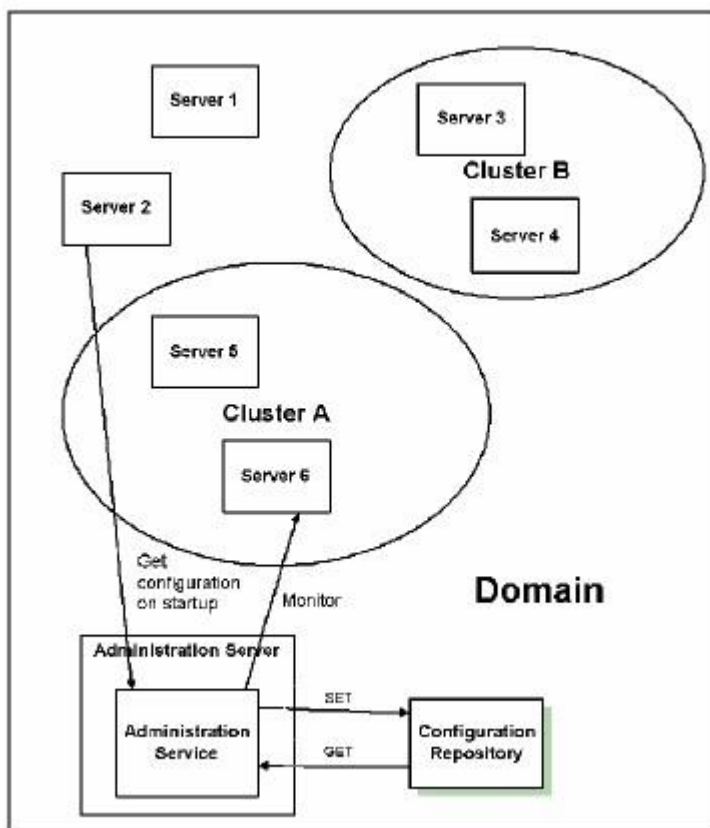
- 队列中空闲线程数。
- 队列中等待时间最长的请求。
- 吞吐量，根据已经处理的请求数来衡量的。

- 队列长度，根据队列中等待请求数来衡量的。
- JVM 堆还有的内存量。

十一、weblogic 集群

10.1 简介

Managed Server 图示



术语解释：

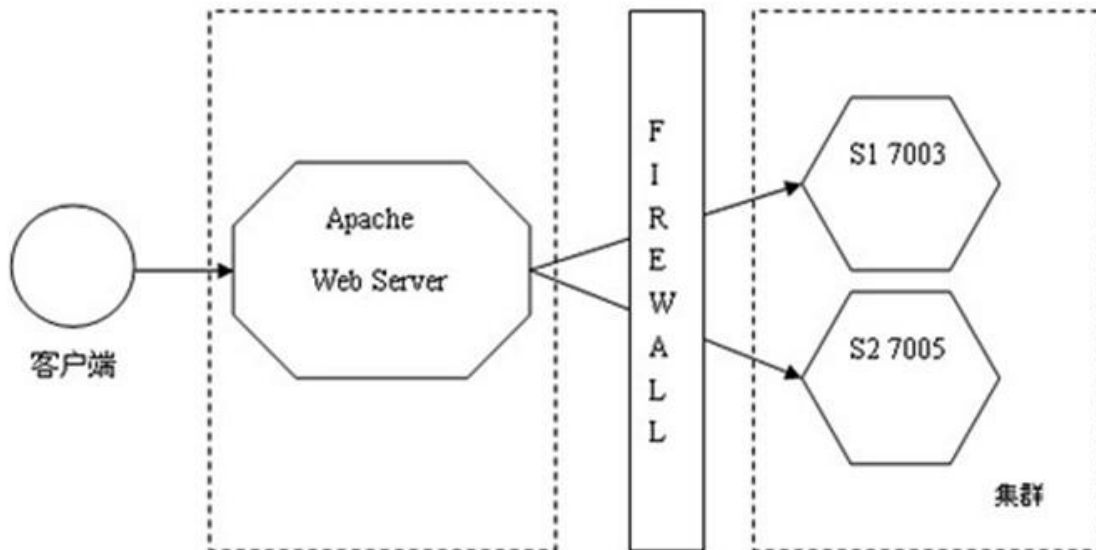
1. Domain 是 WebLogic Server 实例的基本管理单元。由配置为 Administrator Server 的 WebLogic Server 实例管理的逻辑单元，这个单元是所有相关资源的集合。中心配置文件叫 config.xml 。一个域包含一个或多个 WebLogic Server 实例，这些实例可以是群集实例、非群集实例，或者是群集与非群集实例的组合。一个域可以包含多个群集。群集中的所有的服务器实例必须驻留在同一域中；不能将群集“拆分”到多个域中。同样，不能在域之间共享配置的资源或子系统。例如，如果在一个域中创建了 JDBC 连接缓冲池，则不能将其用于另一个域中的服务器实例或群集。（而是必须在另一个域中创建类似的连接缓冲池）。
2. Managed Server，被管理服务器是用来部署运行各种应用程序的。一个域中有一台或多台被管理服务器生产环境中，域由一个管理服务器与多个被管服务器组成。在启动这个域的被管服务器时，首先必须先启动管理服务器，被管服务器启动时，会被命令从管理服务器获得配置信息。这样，管理服务器就成为整个域的配置控制中心。一个域只能有一个活动的管理服务器。被管服务器和管理服务器是多对一的，并且被管服务器由管理

服务器统一管理

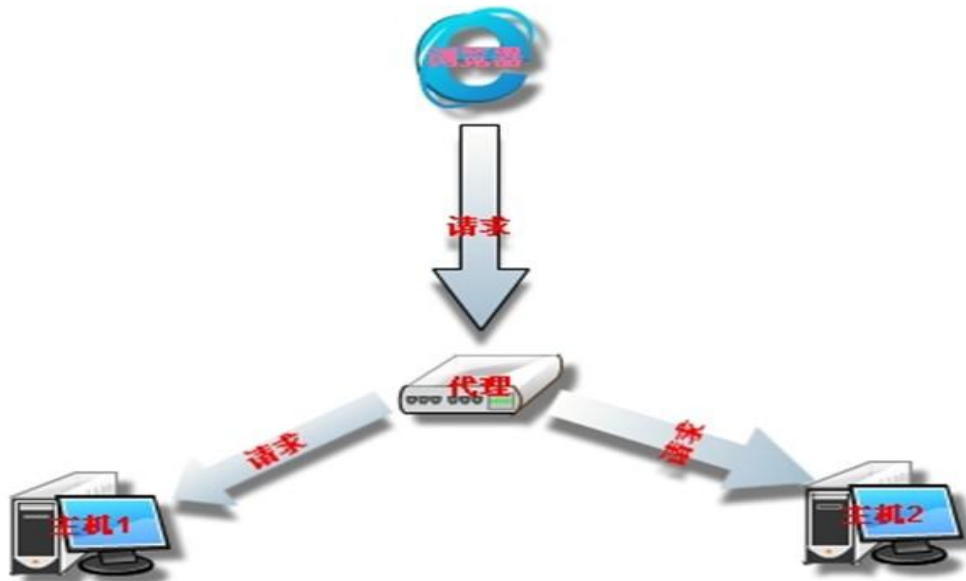
3. **Administrator Server**，管理服务器是用来管理配置域的中心点，一般来说，管理服务器上是不部署应用程序的（应用程序应该部署到被管理服务器上：**Managed Server**），而是用来统一管理、配置、监控被管理服务器以及部署应用程序到被管理服务器上。一个域中有且只有一台管理服务器，管理服务器和域是一一对应的。在每个域中，只有一个 **WebLogic Server** 实例可充当管理服务器：此服务器实例可配置、管理和监视域中所有其他被管理服务器实例和资源。每个管理服务器只管理一个域。如果一个域中包含多个群集，则域中的每个群集都具有相同的管理服务器
4. **Machine**，机器是物理上的概念，代表一台运行 **WebLogic** 应用服务器的实在的机器，包括其 IP 地址等信息。一个域中可以包括多台机器。
5. 代理
 - 集群由代理来实现负载均衡。通过将请求转发到不同的管理服务器上来实现。
 - 代理是用来分发用户请求，代理可以是硬件设备也可以是软件 **Web** 服务器。一般来说硬件代理的性能强大些，而且稳定性也优于软件代理。
 - 硬件带来的厂家主要有：F5 Networks、Radware、array，浪潮、趋势等
 - 软件代理主要有：IIS、Apache、weblogic 等 HTTP 服务器。

10.2.集群结构图

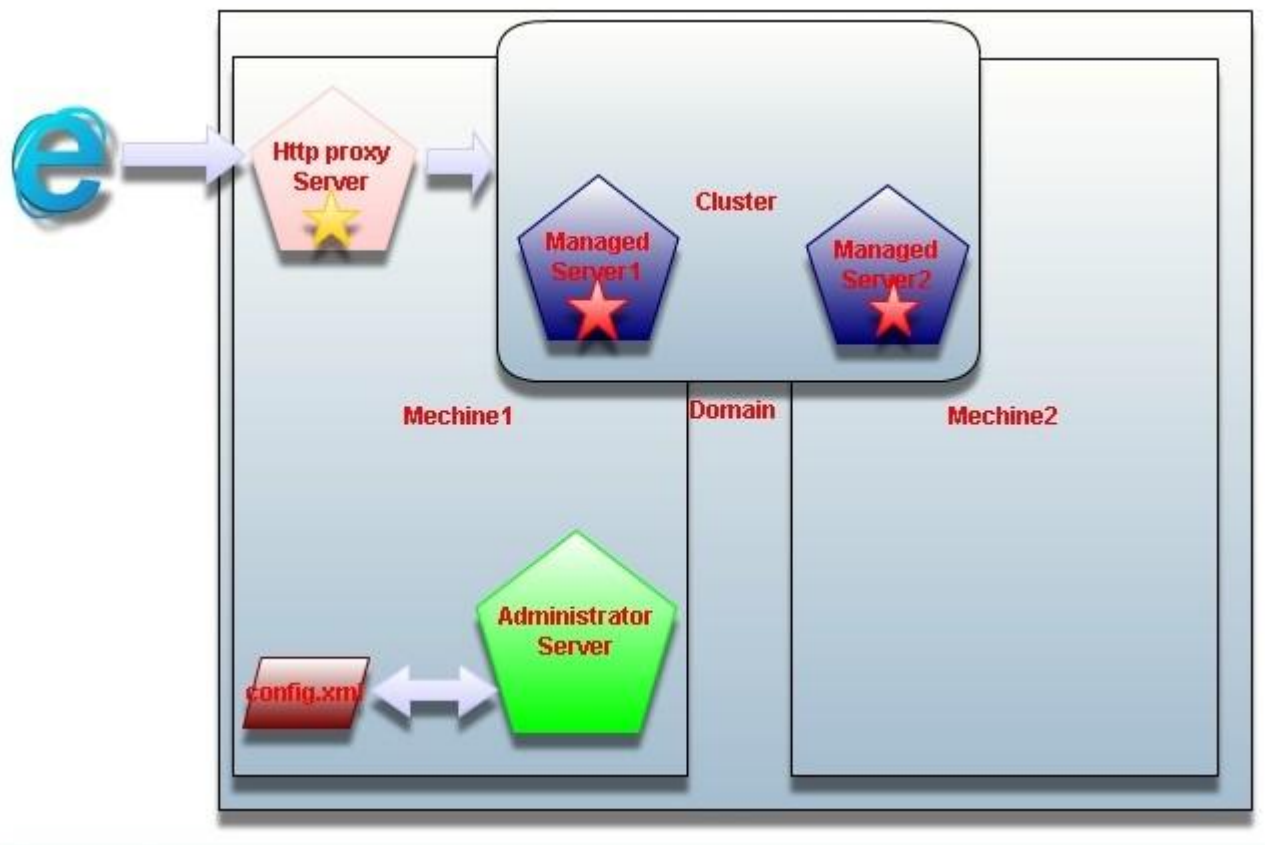
用 **weblogic** 作代理，来配置集群，如果更好的提高性能，可以采用硬件作代理代理示意图：



物理示意图：



逻辑示意图：管理服务器读取配置信息来管理集群中的服务实例

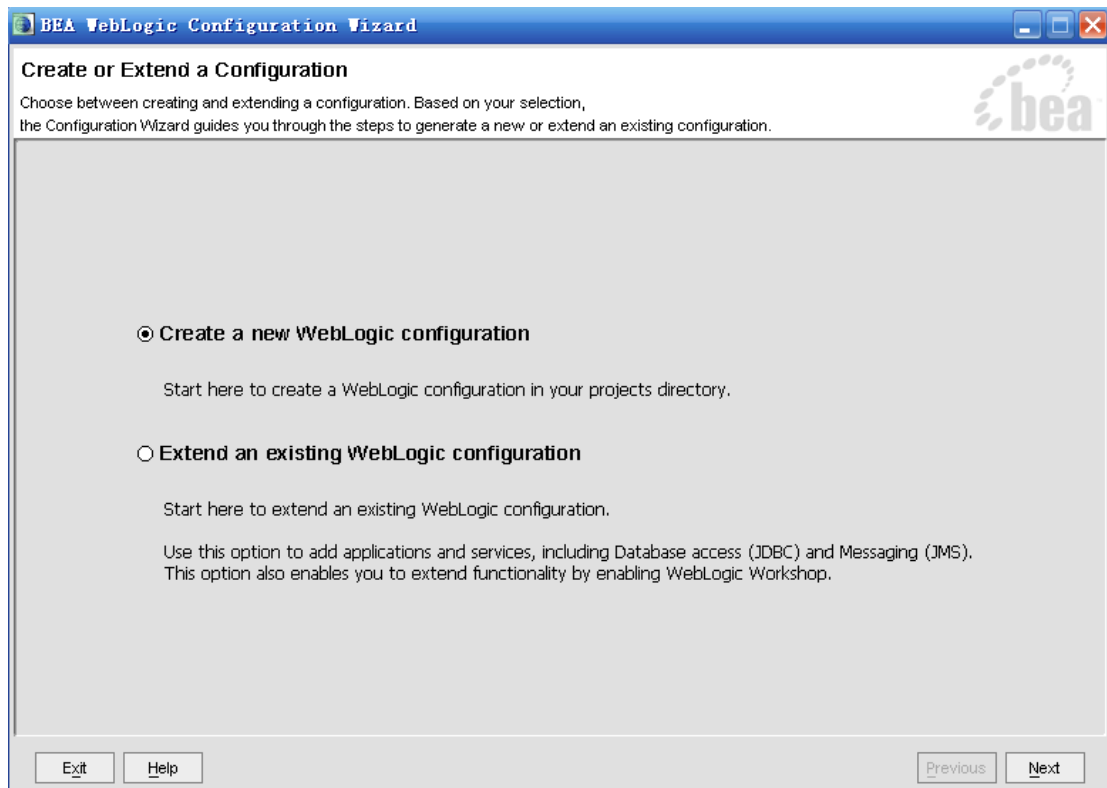


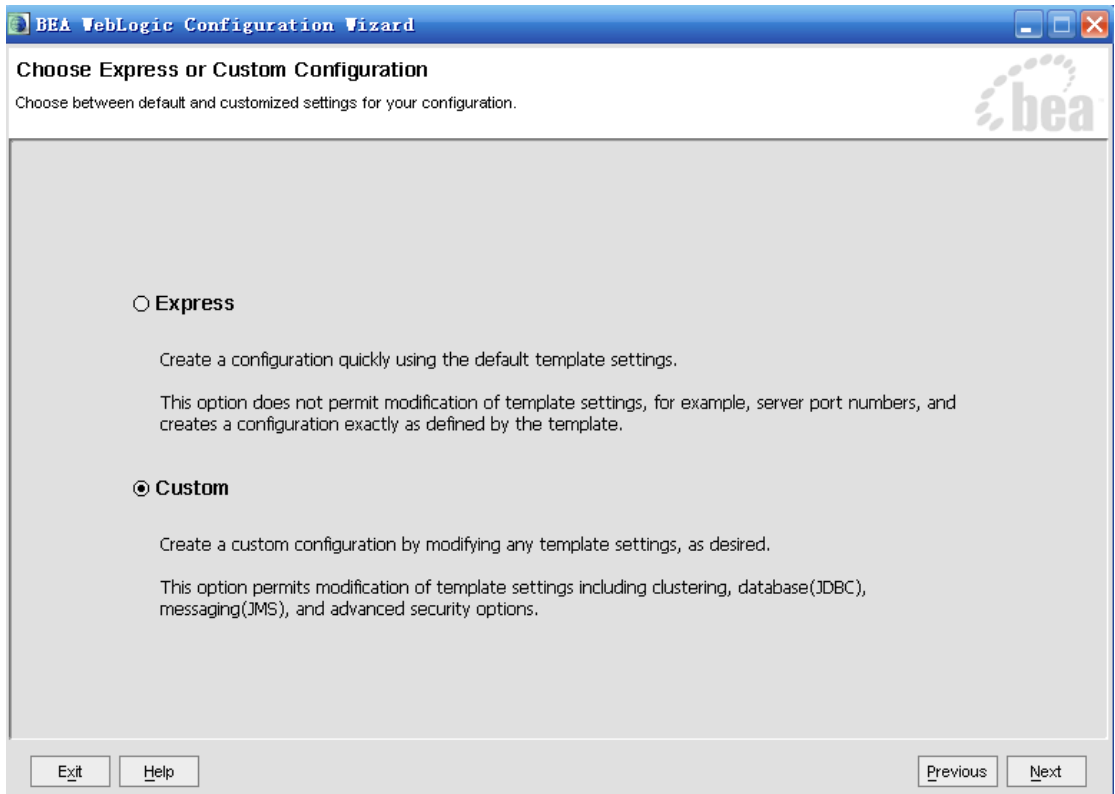
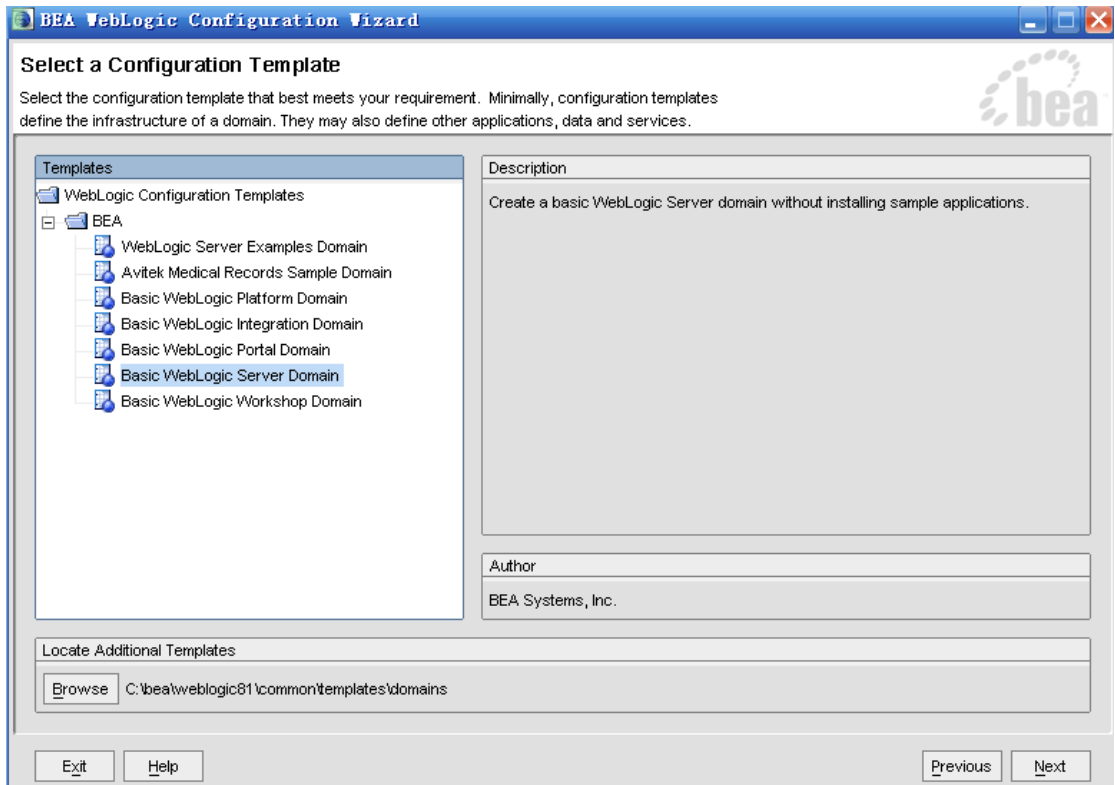
10.3 创建集群

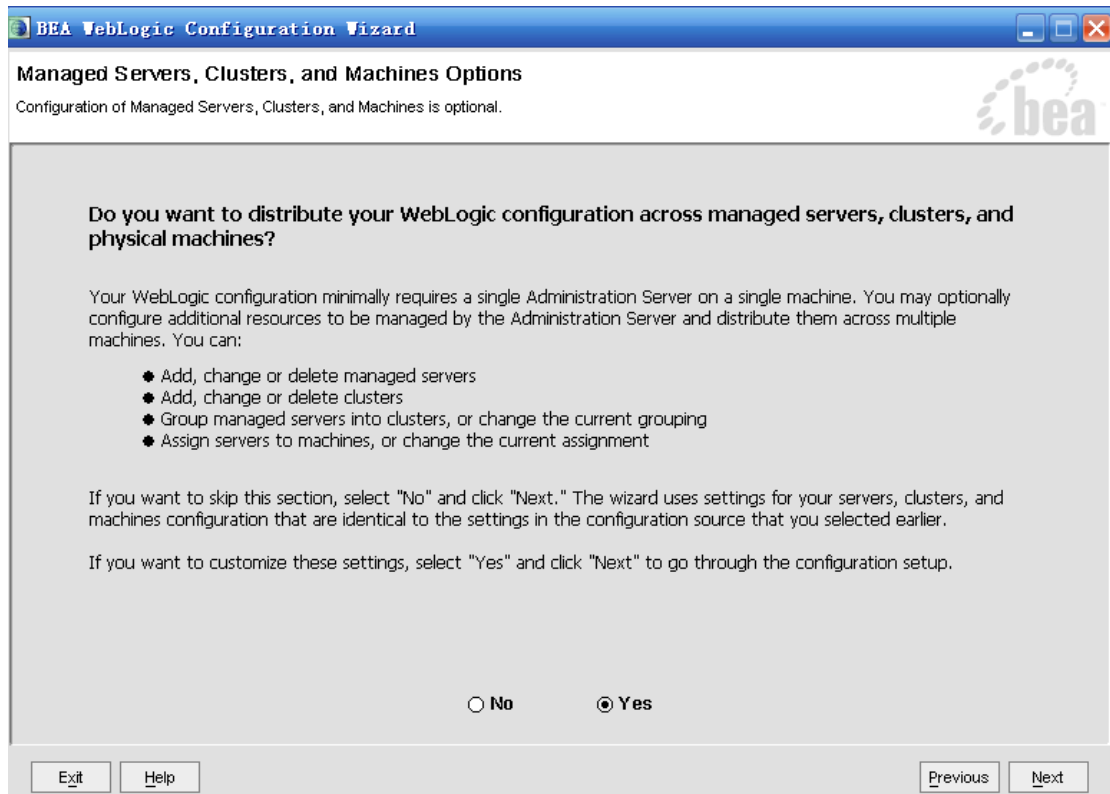
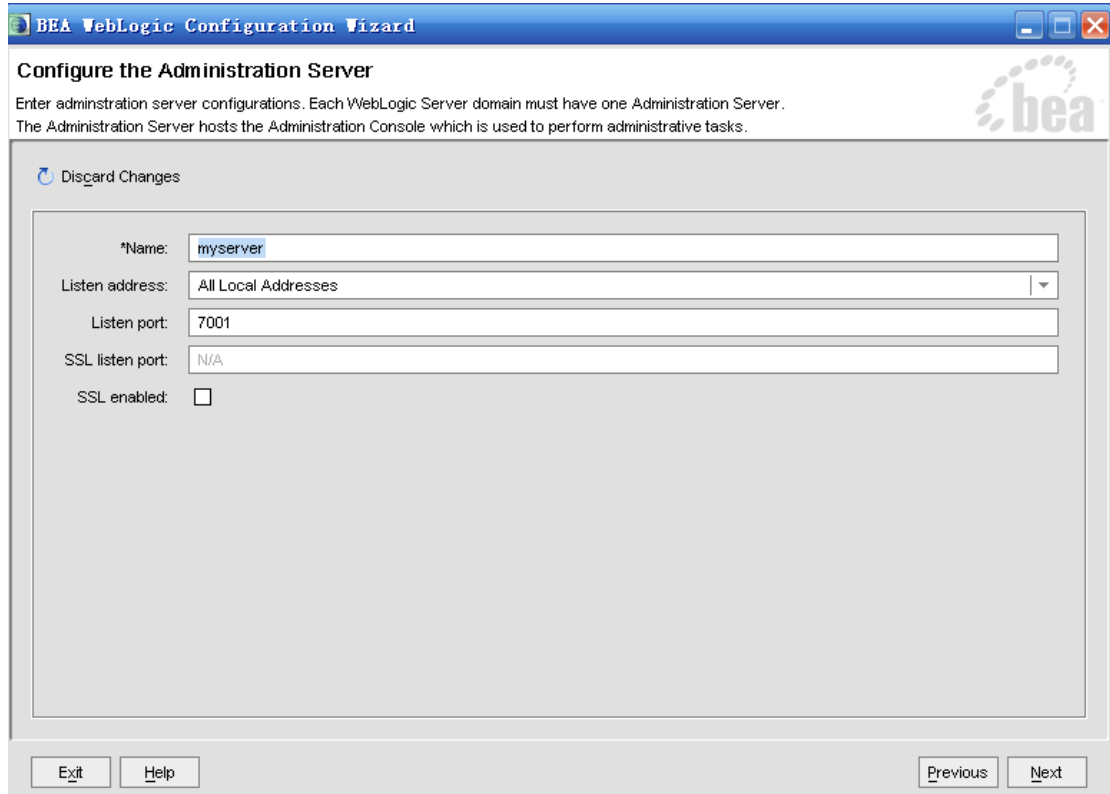
部署表 (IP,PORT)

机器名	机器类型	配置	角色
Machine49	Window XP	192.168.8.49:7001	Administrator Server
Machine49	Window XP	192.168.8.49:7008	Managed Server
Machine49	Window XP	192.168.8.49:7009	Managed Server
Machine49	Window XP	192.168.8.49:7000	Proxy Server
Machine48	Window XP	192.168.8.48:7001	Managed Server

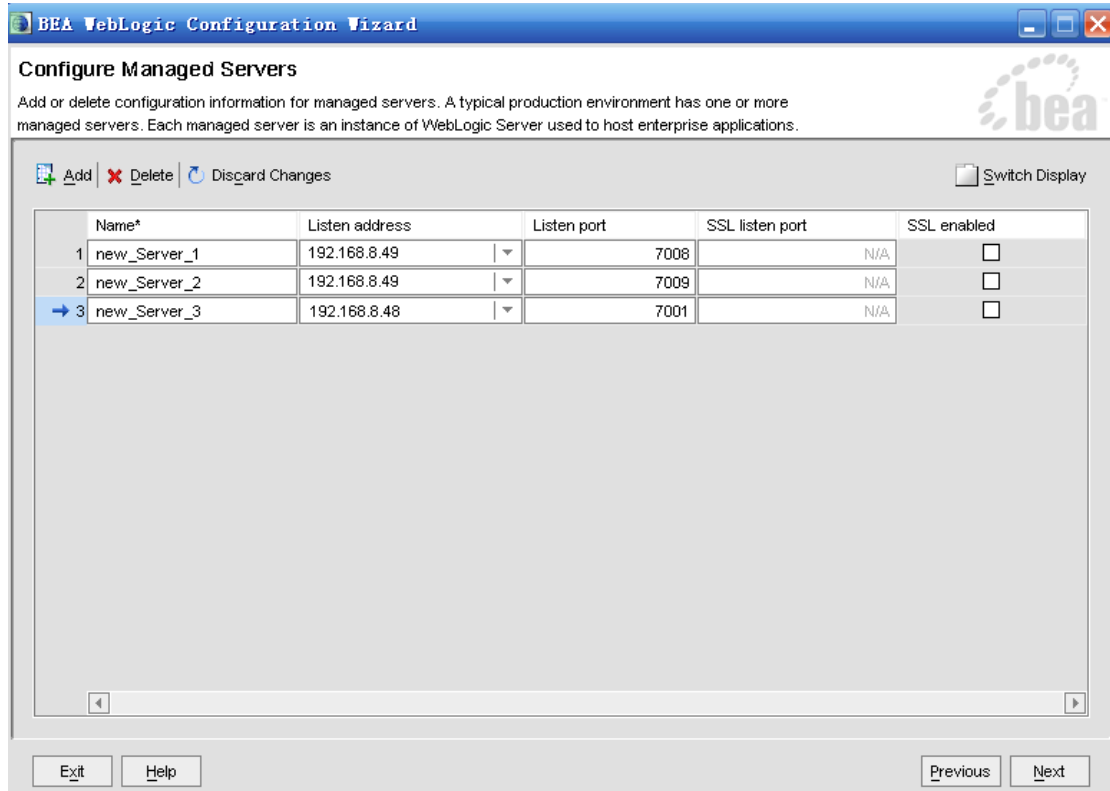
10.3.1 创建管理服务器(Manager Server)



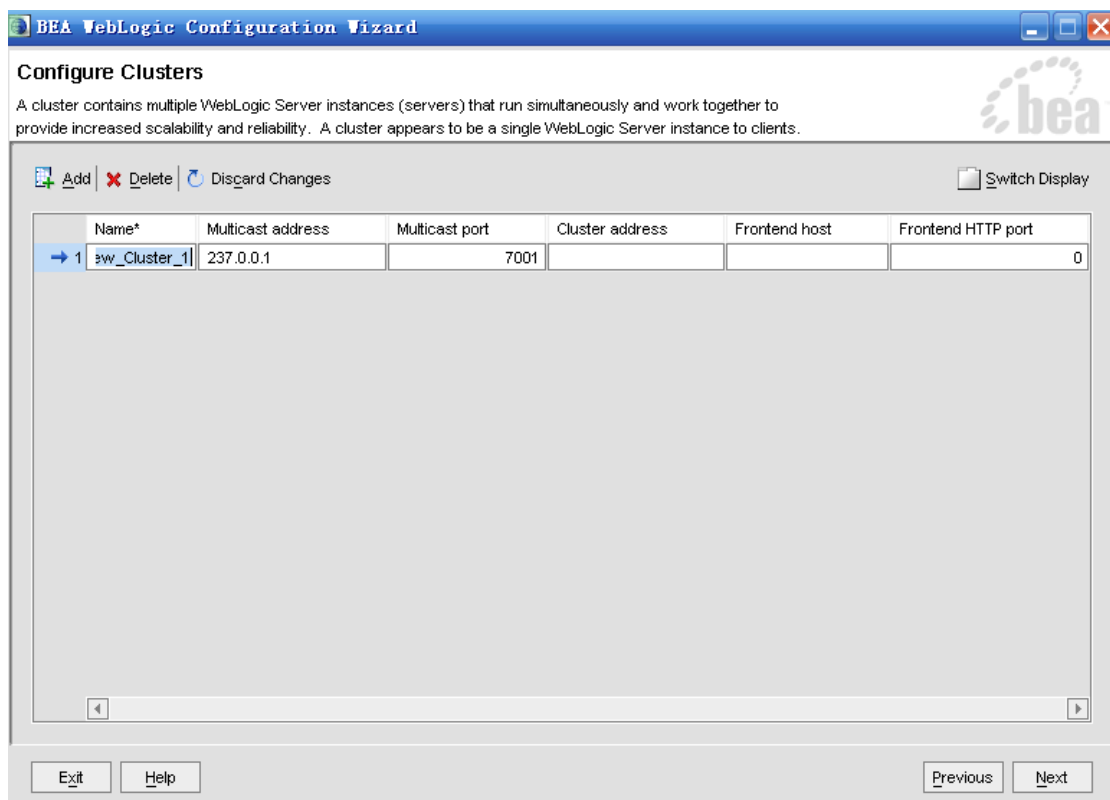




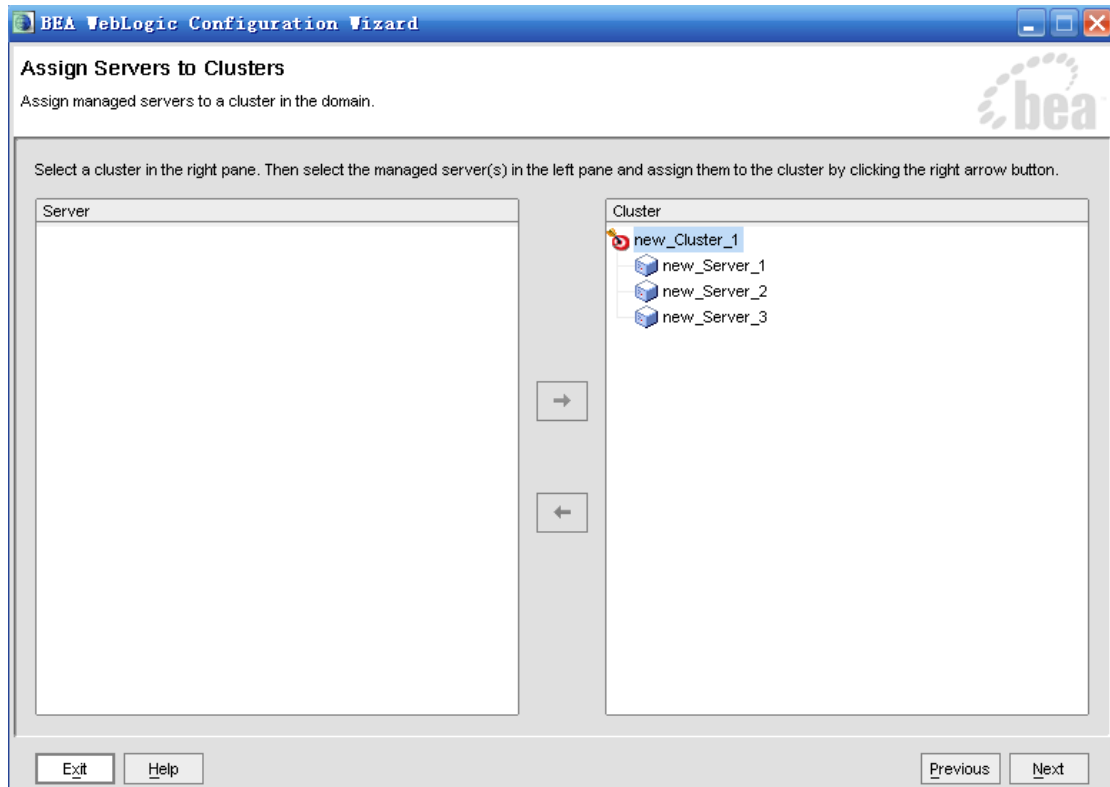
这一步增加被管理服务器



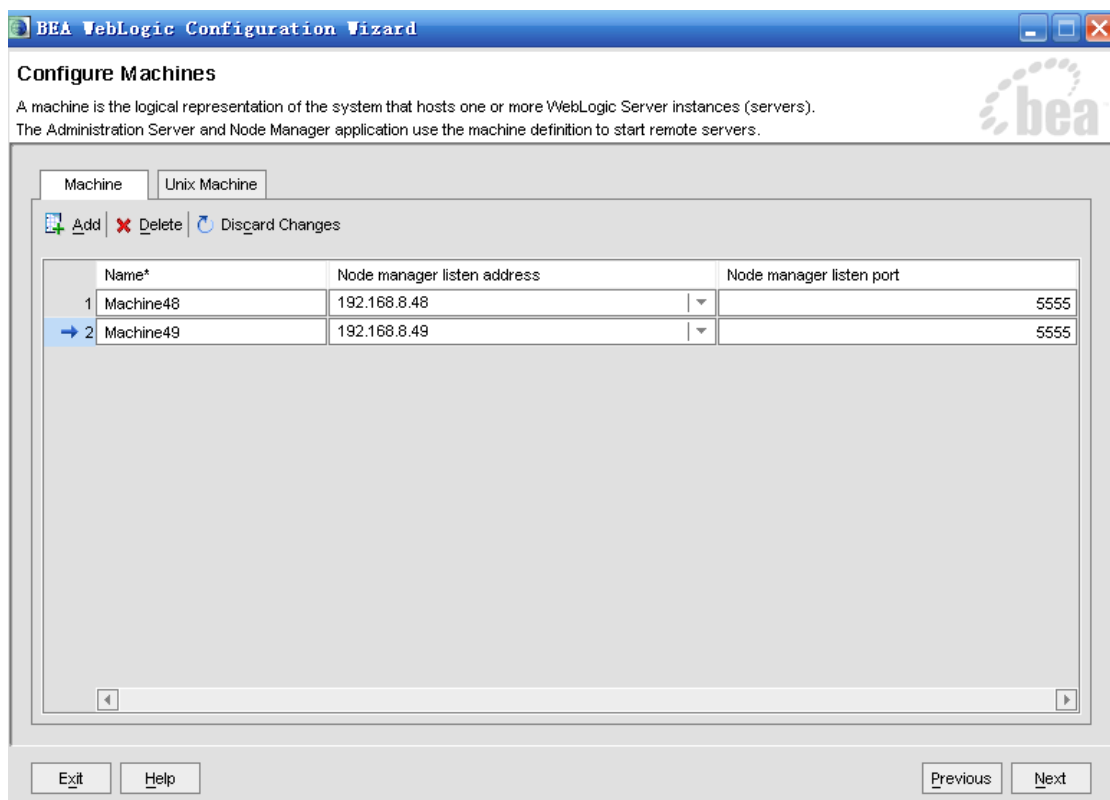
这一步增加集群广播地址和端口，默认即可

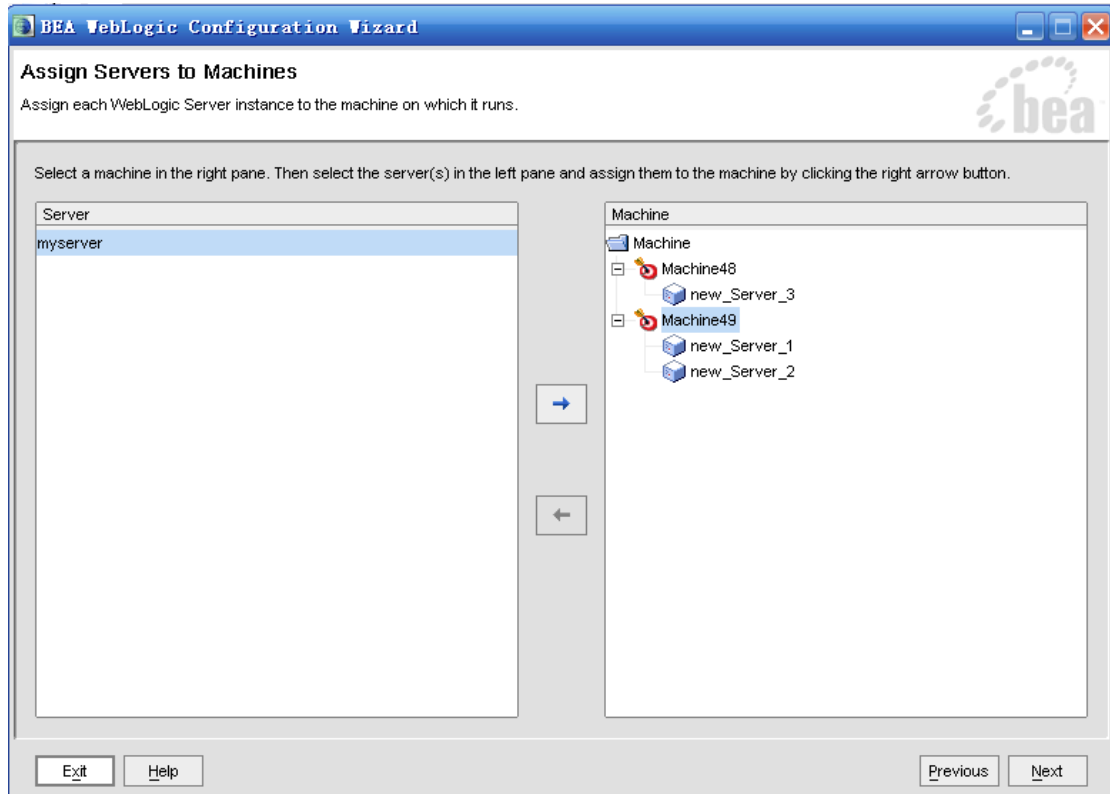


这一步把被管理服务添加到集群中



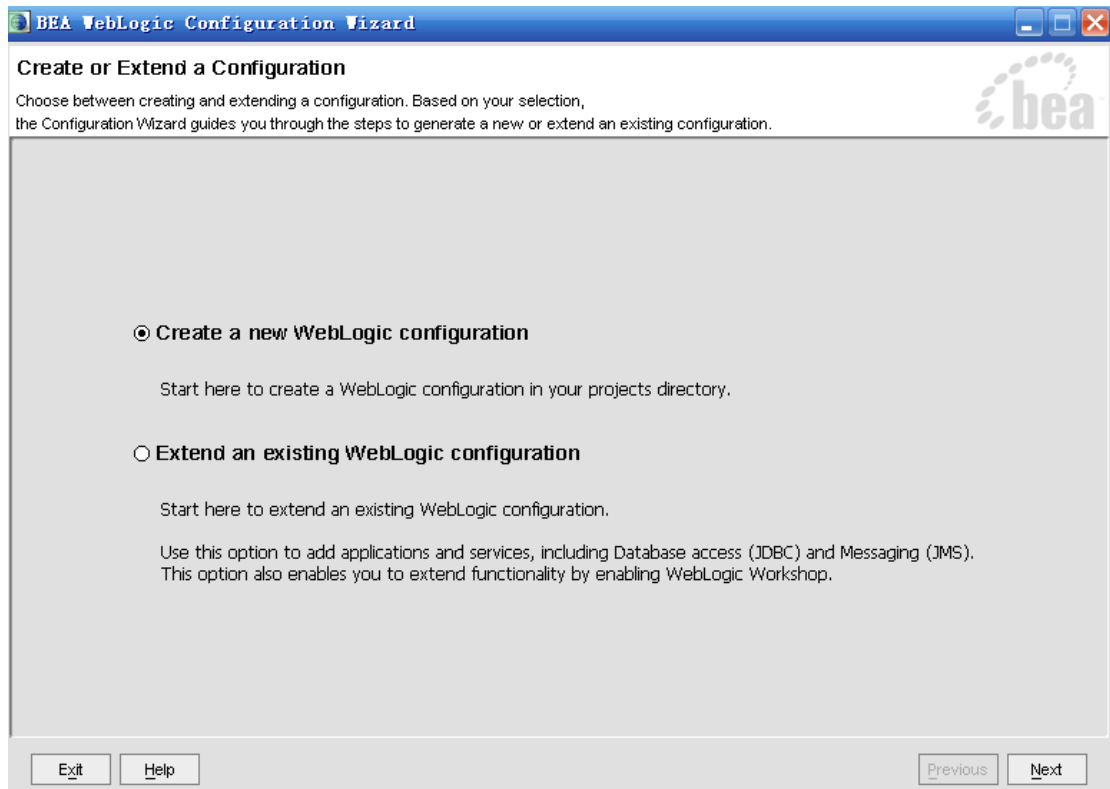
这一步很关键，如果是在同一台机器上，就不用配置这一步了，当有远程机器时，weblogic会通过下面配置的机器来找到绑定的被管理服务器。

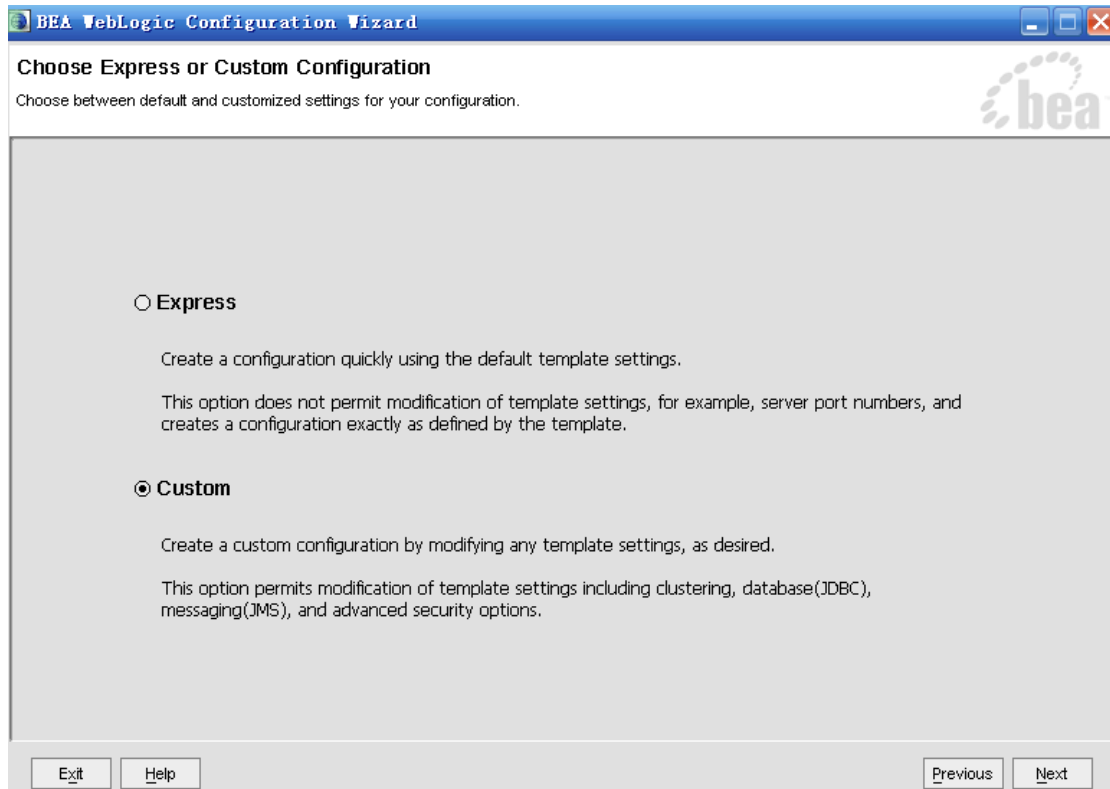
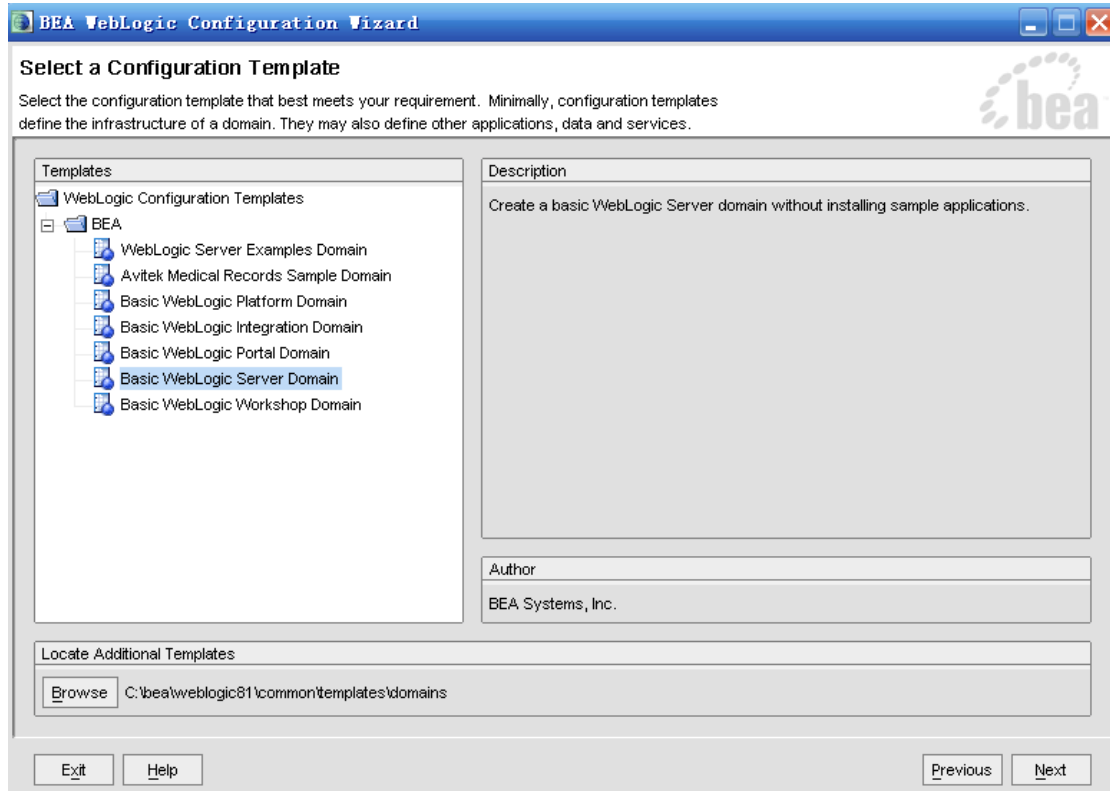


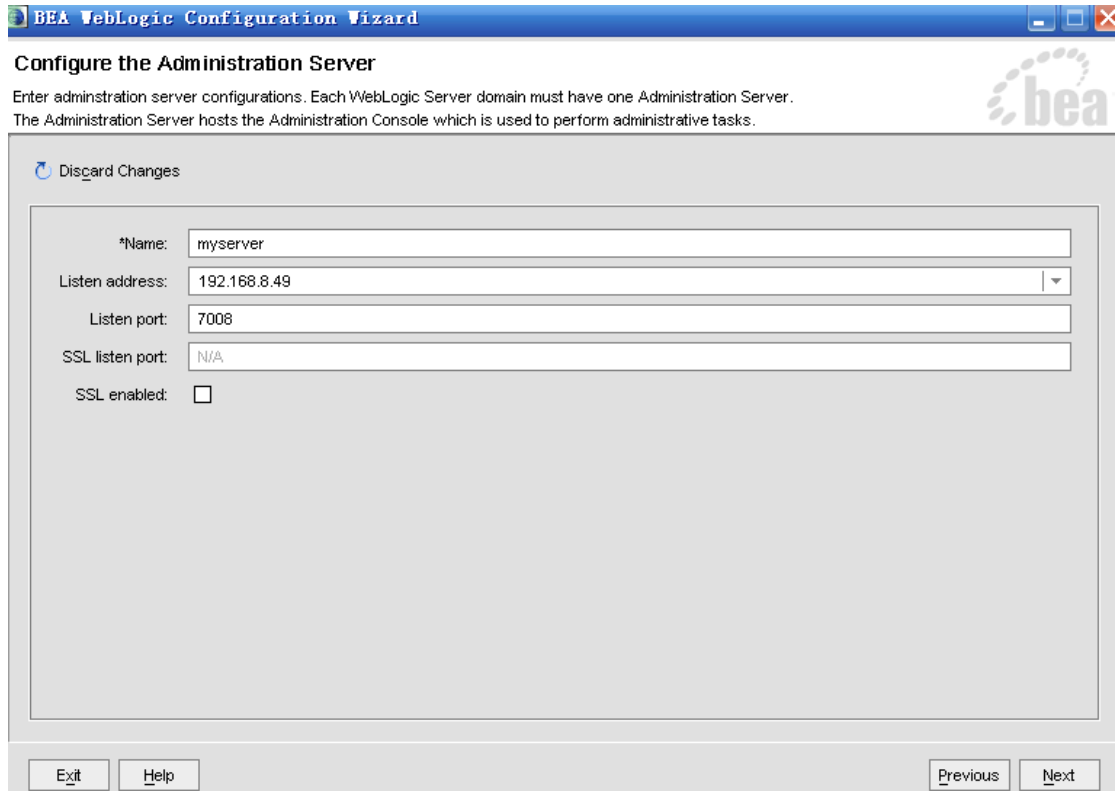


以后步骤都按默认即可。

10.3.2 创建被管理服务器(Managed Server)

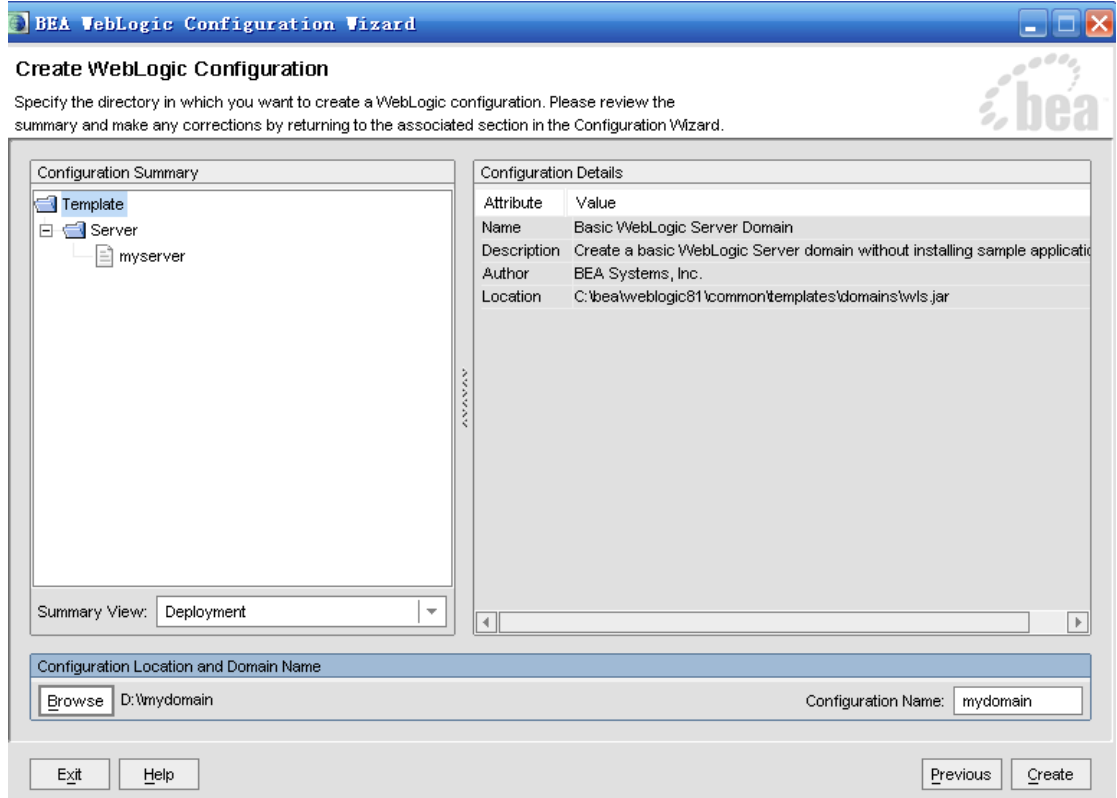






其他步骤默认即可

最后一步，需要注意一个问题，当在同一台机器上配置多个服务时，必须选择不同的目录，千万不能更改域名(默认时 mydomain)，因为集群中的所以机器必须位于同一域中



创建完毕，重复以上步骤。来创建多个被管理服务器

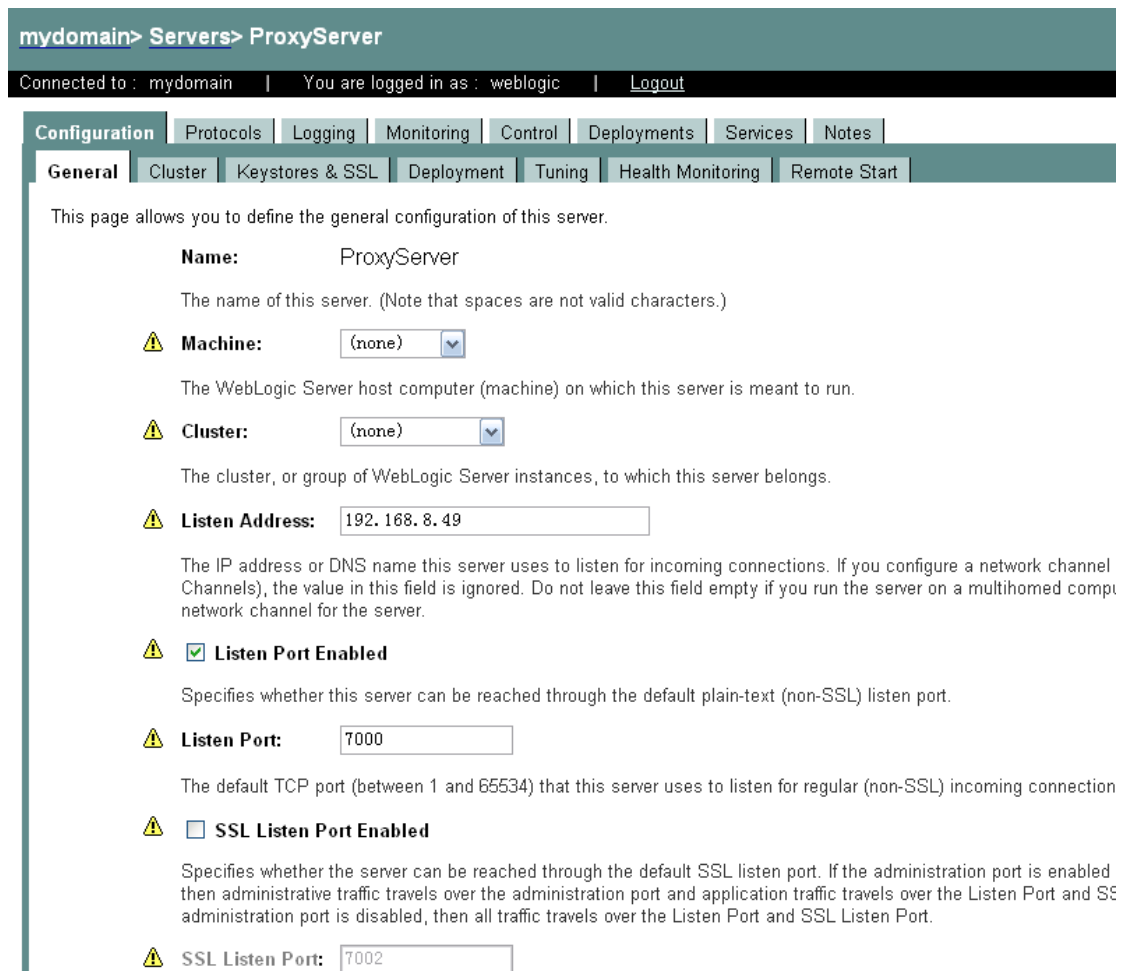
10.3.3 创建 weblogic 代理服务器(proxy Server)

1, 创建一个服务器



The screenshot shows the BEA WebLogic Server Home console. On the left, a tree view shows the 'Servers' folder under 'mydomain'. A context menu is open over the 'Servers' folder, with 'Configure a new Server...' selected. The main content area displays the 'Welcome to BEA WebLogic Server Home' page, which includes sections for 'Information and Resources', 'Domain Configurations', and 'Services Configurations'. The 'Domain Configurations' section lists 'Network Configuration' with links for 'Domain', 'Servers', 'Clusters', and 'Machines'. The 'Services Configurations' section lists 'JDBC' and 'SNMP' with links for 'Connection Pools', 'MultiPools', 'Agent', and 'Proxies'.

2. 可以看出来, 其实 weblogic 的代理服务器, 就相当于一个被管理服务器, 所以启动方式和被管理的启动方式一样



The screenshot shows the configuration page for a ProxyServer in the BEA WebLogic Server console. The breadcrumb path is 'mydomain > Servers > ProxyServer'. The page title is 'Configuration' and the sub-tab is 'General'. The page content includes the following fields and options:

- Name:** ProxyServer
The name of this server. (Note that spaces are not valid characters.)
- Machine:** (none) [v]
The WebLogic Server host computer (machine) on which this server is meant to run.
- Cluster:** (none) [v]
The cluster, or group of WebLogic Server instances, to which this server belongs.
- Listen Address:** 192.168.8.49
The IP address or DNS name this server uses to listen for incoming connections. If you configure a network channel (Channels), the value in this field is ignored. Do not leave this field empty if you run the server on a multihomed computer network channel for the server.
- Listen Port Enabled:** Listen Port Enabled
Specifies whether this server can be reached through the default plain-text (non-SSL) listen port.
- Listen Port:** 7000
The default TCP port (between 1 and 65534) that this server uses to listen for regular (non-SSL) incoming connection
- SSL Listen Port Enabled:** SSL Listen Port Enabled
Specifies whether the server can be reached through the default SSL listen port. If the administration port is enabled, then administrative traffic travels over the administration port and application traffic travels over the Listen Port and SSL administration port is disabled, then all traffic travels over the Listen Port and SSL Listen Port.
- SSL Listen Port:** 7002

3. 创建一个 web 工程 proxy，里面只包含 web.xml 和 weblogic.xml

```
#####web.xml#####
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
```

```
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
```

```
<servlet>
```

```
  <servlet-name>HttpClusterServlet</servlet-name>
```

```
  <servlet-class>
```

```
    weblogic.servlet.proxy.HttpClusterServlet
```

```
</servlet-class>
```

```
<init-param>
```

```
  <param-name>WebLogicCluster</param-name>
```

```
  <param-value>
```

```
    <!--集群中的被管理服务器-->
```

```
    192.168.8.49:7008|192.168.8.49:7009|192.168.8.48:7001
```

```
  </param-value>
```

```
</init-param>
```

```
<init-param>
```

```
  <param-name>DebugConfigInfo</param-name>
```

```
  <param-value>ON</param-value>
```

```
</init-param>
```

```
<init-param>
```

```
  <param-name>verbose</param-name>
```

```
  <param-value>>true</param-value>
```

```
</init-param>
```

```
</servlet>
```

```
<servlet-mapping>
```

```
  <servlet-name>HttpClusterServlet</servlet-name>
```

```
  <url-pattern>/</url-pattern>
```

```
</servlet-mapping>
```

```
<servlet-mapping>
```

```
  <servlet-name>HttpClusterServlet</servlet-name>
```

```
  <url-pattern>*.jsp</url-pattern>
```

```
</servlet-mapping>
```

```
<servlet-mapping>
```

```
  <servlet-name>HttpClusterServlet</servlet-name>
```

```
  <url-pattern>*.htm</url-pattern>
```

```
</servlet-mapping>
```

```
<servlet-mapping>
```

```
  <servlet-name>HttpClusterServlet</servlet-name>
```

```
  <url-pattern>*.html</url-pattern>
```

```
</servlet-mapping>
```

```

<servlet-mapping>
  <servlet-name>HttpClusterServlet</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>

#####weblogic.xml#####
<weblogic-web-app>
  <container-descriptor>
    <prefer-web-inf-classes>true</prefer-web-inf-classes>
  </container-descriptor>
  <context-root></context-root>
</weblogic-web-app>

```

创建好后，把它拷贝到管理服务器的 applications 目录下。把它部署到代理服务器上

The screenshot shows the 'Configuration' tab with sub-tabs for 'Targets', 'Deploy', 'Monitoring', 'Testing', and 'Notes'. Below the tabs is a text block explaining that this page allows defining servers or clusters for deployment. The main content area is divided into two sections: 'Independent Servers' and 'Clusters'. In 'Independent Servers', 'myserver' is unchecked and 'ProxyServer' is checked. In 'Clusters', 'new_Cluster_1' is selected with radio buttons for 'All servers in the cluster' and 'Part of the cluster'. Under 'Part of the cluster', three server checkboxes are shown: 'new_Server_1', 'new_Server_2', and 'new_Server_3'. An 'Apply' button is visible at the bottom right.

测试代理应用是否成功

http://192.168.8.49:7000/xxx.jsp? [WebLogicBridgeConfig](#)

10.3.5 用 Apache 作代理服务器(推荐)

上面是用 weblogic 作代理服务器，也可以用 apache 作代理服务器，配置相当简单一些

1. 下载并安装 apache 服务器 www.apache.org，经过测试发现不能用 2.2.x 版本
参考文档：

http://download.oracle.com/docs/cd/E13222_01/wls/docs81/plugins/apache.html#113634

2. 安装 apache 代理插件

在 weblogic 安装目录下面，找到 bea/weblogic81/server/bin 目录，下面有两个文件：

mod_wl_20.so, mod_wl128_20.so 分别是对应不同版本的 apache, 这里用来测试的是 2.0 版的 apache, 所以使用了第一个文件。把 mod_wl_20.so 这个文件 copy 到 apache 安装目录下的 modules 目录中

配置 apache 的 httpd.conf, 加入以下语句:

```
#表示在启动apache的时候加载weblogic的插件
LoadModule weblogic_module modules/mod_wl_20.so
<IfModule mod_weblogic.c>
#表示集群的各个成员地址
WebLogicCluster 192.168.8.49:7008,192.168.8.49:7009,192.168.8.48:7001
  MatchExpression *.jsp
  MatchExpression *.do
</IfModule>
#监听地址和端口
#Listen 12.34.56.78:80
Listen 80
#主页
DirectoryIndex index.jsp index.html.var
```

3. 把 web 工程部署到 weblogic 上, 具体参考上面部署, 测试 <http://192.168.8.49/web/index.jsp>

10.3.6 启动服务

1. 启动管理服务器

进入 C:\bea\user_projects\domains\mydomain 目录, 执行 startWebLogic.cmd 启动

2. 启动被管理服务器

在每台被管理服务器机器上, 进入 C:\bea\user_projects\domains\mydomain

```
# startManagedWebLogic [被管理服务名称] [管理服务器的地址]
```

执行 startManagedWebLogic.cmd new_Server_3 <http://192.168.8.49:7001>

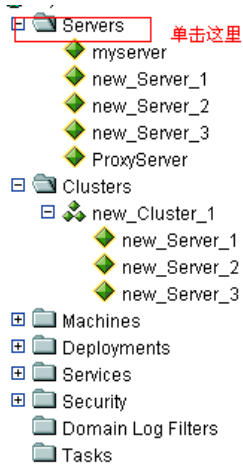
执行 startManagedWebLogic.cmd new_Server_1 <http://192.168.8.49:7001>

3. 启动代理服务器

进入管理服务器所在的机器, 进入 C:\bea\user_projects\domains\mydomain 目录

```
执行 startManagedWebLogic.cmd ProxyServer http://192.168.8.49:7001
```

如果都启动成功, 和下面的状态一样, 都是 running



Connected to : mydomain | You are logged in as : weblogic | [Logout](#)

A server is an instance of WebLogic Server that runs in its own Java Virtual Machine (JVM) server that acts as the Administration Server. In a typical production environment, the Administration Server is called myserver. A typical production environment uses several instances of WebLogic Server used to host enterprise applications.

This Servers page displays key information about each server that has been configured.

[Configure a new Server...](#)

[Customize this view...](#)

Name	Listen Port	Listen Port Enabled	State	
myserver	7001	true	RUNNING	
new_Server_1	7008	true	RUNNING	
new_Server_2	7009	true	RUNNING	
new_Server_3	7001	true	RUNNING	
ProxyServer	7000	true	RUNNING	

10.3.7 测试

1. 创建一个 web 工程,名称为 web, 创建一个 jsp 页面。

```
<%
    out.println("OK");

    System.out.println("OK");

    if (session.getAttribute("session name") == null) {

        session.setAttribute("session name", "session value");

        out.println("session value is null ,set it "
            + session.getAttribute("session name"));

    } else {

        out.println("session value is set : "
            + session.getAttribute("session name").toString());

    }
%>
```

2. 如何 Session 复制

由于集群环境中, 用户访问的请求在不同的被管理服务器之间不停的切换, 而用户访问又需要保持状态 (Session), 这就要求 Session 可以穿梭于被管服务器之

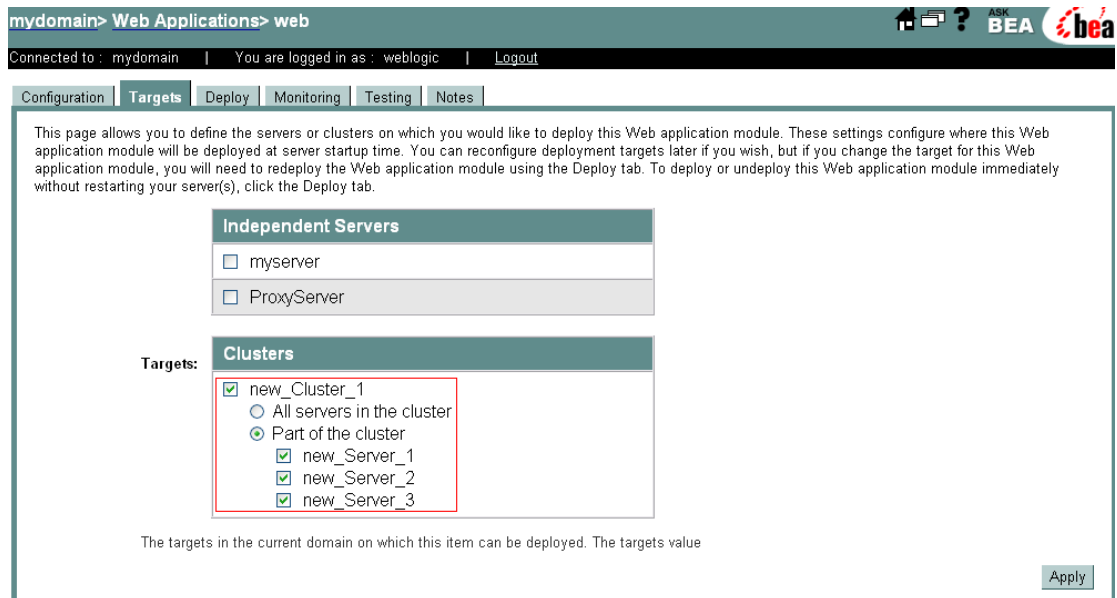
间，就是各个被管服务器上的 Session 是一致的，这样用户才感觉不到请求的切换。

创建一个 weblogic.xml

```
<weblogic-web-app>
  <session-descriptor>
    <session-param>
      <param-name>PersistentStoreType</param-name>
      <param-value>replicated</param-value>
    </session-param>
  </session-descriptor>
</weblogic-web-app>
```

3. 部署 web 工程

把它拷贝到管理服务器的 applications 目录下，进入控制台，选择部署一个新的 web 工程



4、测试

访问被代理服务器

<http://192.168.8.49:7008/web/>

http://192.168.8.49:7008/web/

http://192.168.8.48:7001/web/

访问代理服务器

http://192.168.8.49:7000/web/

10.3.8 负载均衡

通过 Apache 中所带的 ab 包来进行并发访问的模拟测试，压力测试。

ab -n 100 -c 10 http://192.168.8.49 /web/index.jsp

- ab 是测试程序的名称
- 参数 n 代表请求的总数量
- 参数 c 代表并发的请求数

- url 为要测试压力的页面

注：使用这个命令时，一定要在系统路径中能够找到该程序，否则不能执行。

```
CMD> ab -n 100 -c 10 http://192.168.8.49 /web/index.jsp
```

```
Server Software:      Apache/2.0.63
Server Hostname:      192.168.8.49
Server Port:          80
```

```
Document Path:        /web/index.jsp
Document Length:      631 bytes
```

```
Concurrency Level:    10 #当前并发数
Time taken for tests:  36.46875 seconds
Complete requests:    1000 #总请求数
Failed requests:      0
Write errors:         0
Total transferred:    905334 bytes
HTML transferred:    631000 bytes
Requests per second:  27.74 [# /sec] (mean)
Time per request:     360.469 [ms] (mean)
#平均每个并发数的请求时间
Time per request:     36.047 [ms] (mean, across all concurrent requests)
Transfer rate:        24.52 [Kbytes/sec] received
```

#网络上消耗的时间的分解，各项数据的具体算法还不是很清楚

```
Connection Times (ms)
              min  mean[+/-sd] median  max
Connect:     0    0  3.3    0    31
Processing:   0  358 2734.7   46  28250
Waiting:     0  357 2734.7   46  28250
Total:       0  359 2734.7   46  28250
```

Percentage of the requests served within a certain time (ms)

```
50%    46 #就是有50%的请求都是在46ms内完成的
66%    62
75%    78
80%    93
90%   187
95%   343
98%   687
99%  27125
100% 28250 (longest request)
```

测试结果：经过用 weblogic 代理和 apache 代理的测试，发现当请求数比较少时，weblogic 的代理性能比较好一些，但当请求数特别大时，apache 的优势就比较明显了。所以推荐采用

apache 作代理服务器。