提高WebLogic应用性能的生命周期

http://helloweblogic.com

http://fm928.blog.163.com

个人简介

- 郑波涛 （helloweblogic）
- 北京东方龙马 华东区中间件负责人
- 浙江Oracle User Group 联合创始人
- 博客：http://fm928.blog.163.com http://helloweblogic.com
- QQ：155282417
- TEL /微信：13456782862
- MAIL：zhengbt@olm.com.cn

ZJOUG

China中间件

内容简介

- 优化性能应该贯穿WebLogic应用的整个生命周期，性能规划、实现、测试到部署和运维的生命周期，各个环节都需要考虑性能或者处理性能的问题。

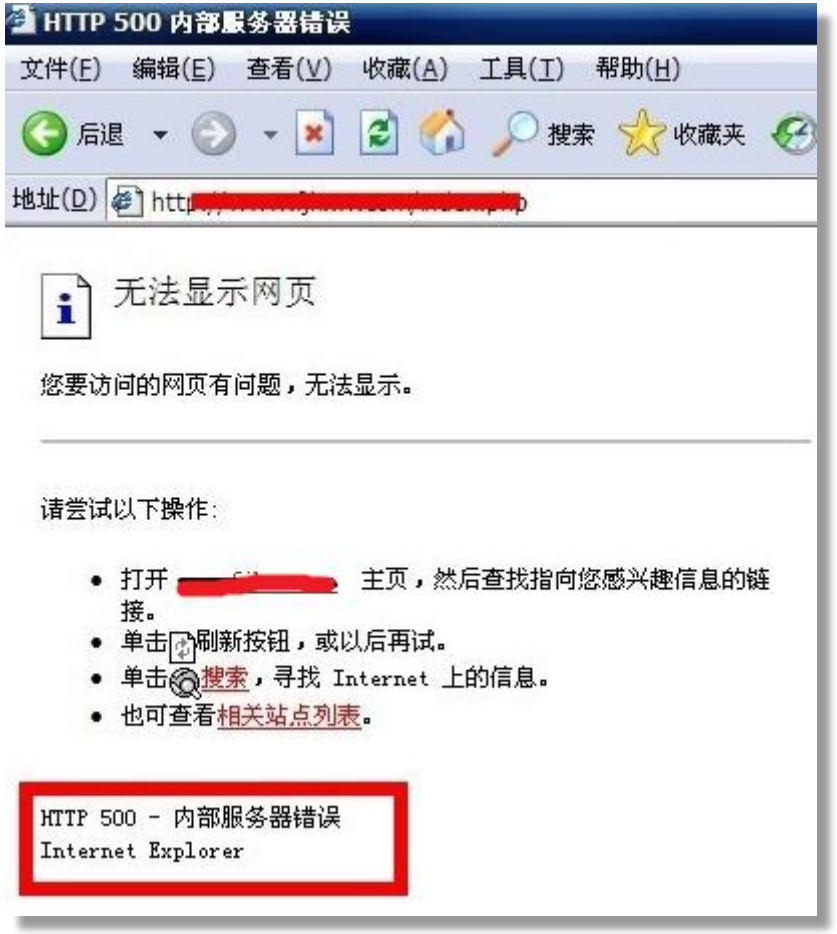
- 本次分享主要是从技术层面进行分享应用性能问题的分析和调整思路。介绍一些具体的提高系统性能和可靠性的技术实现。

性能问题 --- 从一个场景故事开始

- 李强东是一个钓鱼爱好者，而且有商业头脑
- 创办了一个渔具网站和论坛，出售渔具
- 3年时间用户增长迅速，每天上千订单
- 辉煌前景就在眼前，就在这时… …

性能问题 --- 从一个场景故事开始

- 性能问题出现了
- 页面响应时间非常慢，几十秒。。。
- 交易经常报错。。。
- 甚至订单丢失。。。
- CPU 使用率经常100%。。。
- 内存使用率90%。。。



性能问题 --- 从一个场景故事开始

- 李总决定立刻解决这个问题，但是没有头绪…
- 于是李总找了一家第三方的专业集成商
- 将网站系统性能指标写入需求说明书

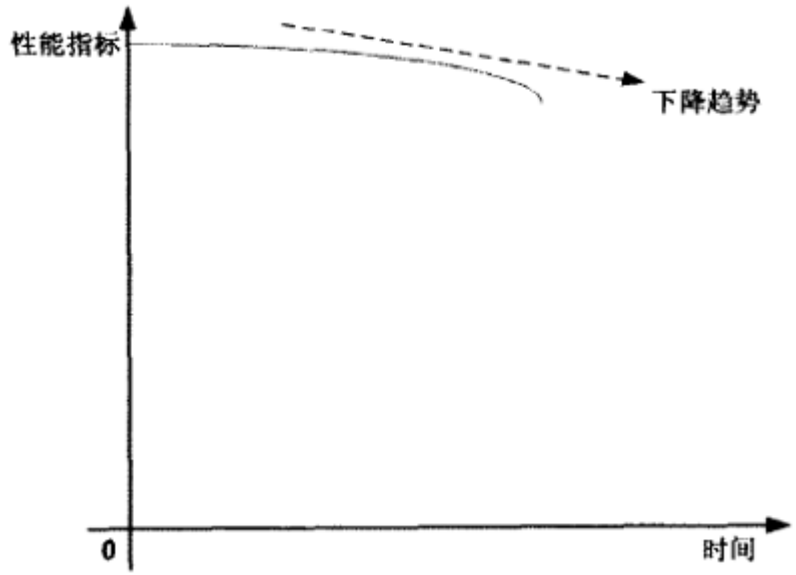
I. 系统高峰期页面的评价响应时间<2s
II. 系统每天至少处理订单2000个
III. 满足5年内每年业务量增加50%的负载需求
IV. 机器资源使用率不超过80%
V. 并发客户端数支持10000个
VI. … …

性能问题 --- 从一个场景故事开始

这就是典型的性能问题，
我们作为第三方怎么来做这个新的网站？

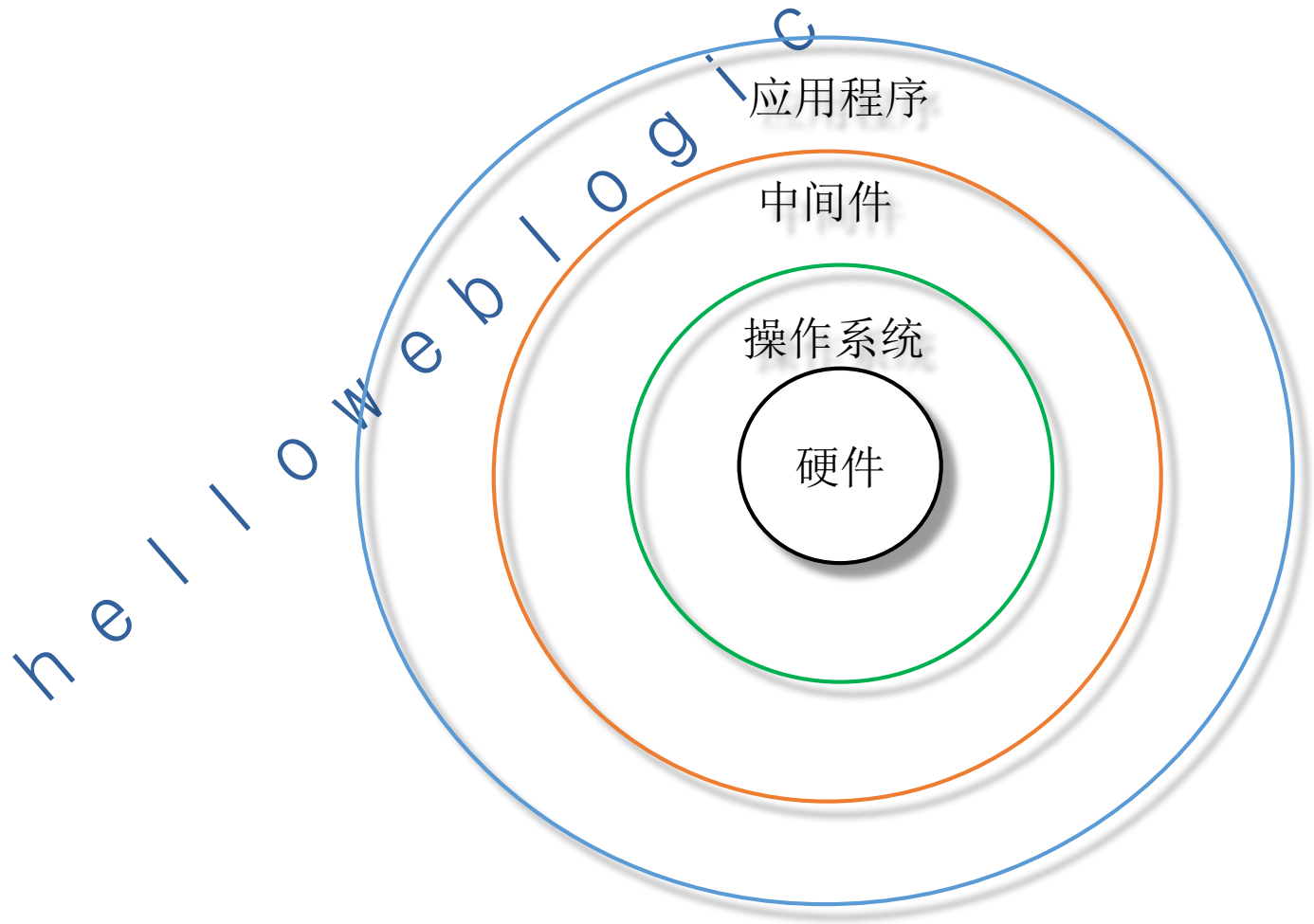
性能问题的现象

- 交易出错
- 交易速度慢
- 资源使用问题
- 性能下降问题



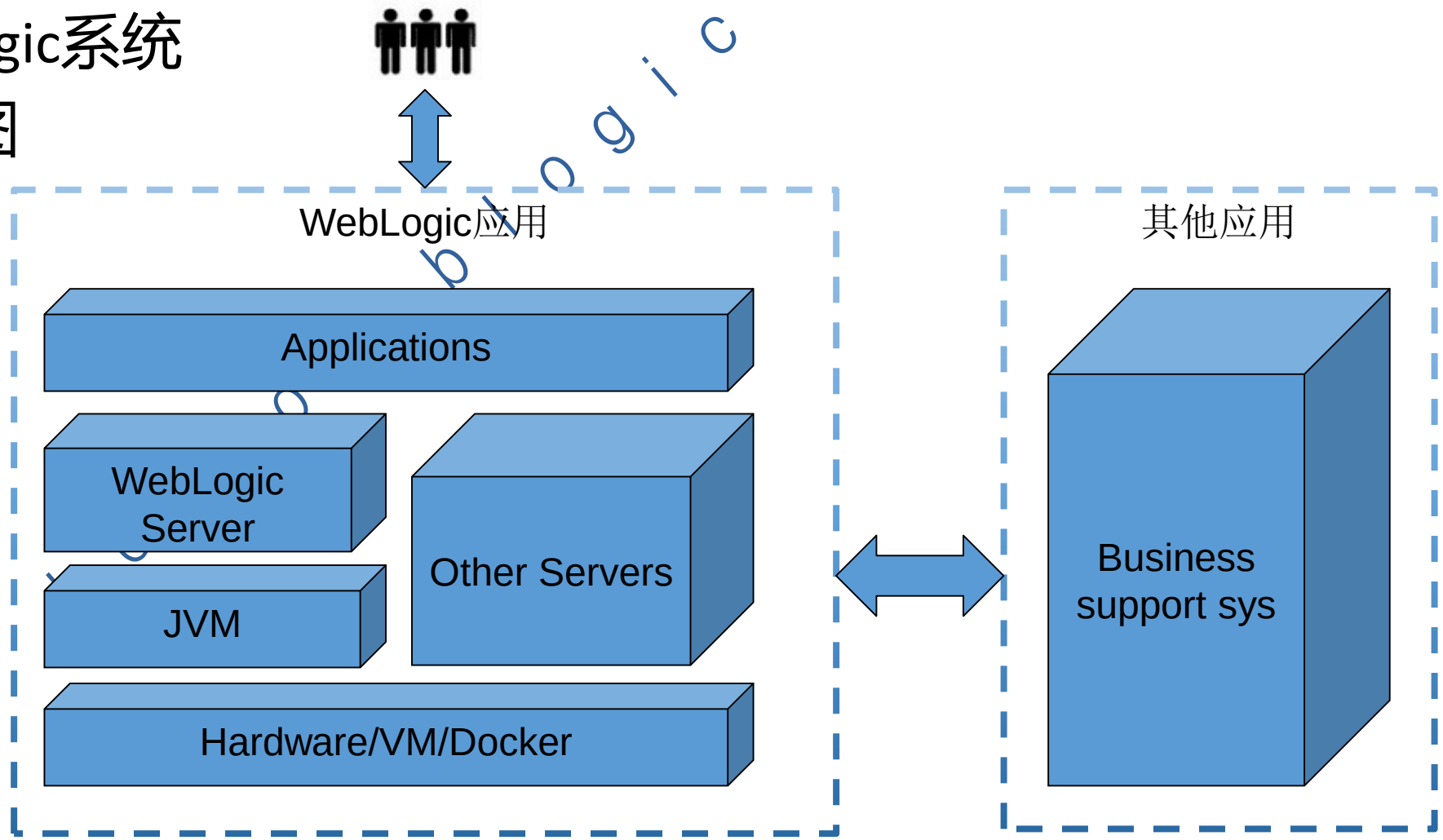
WebLogic应用

- 顾名思义



WebLogic应用性能因素

- WebLogic系统概念图



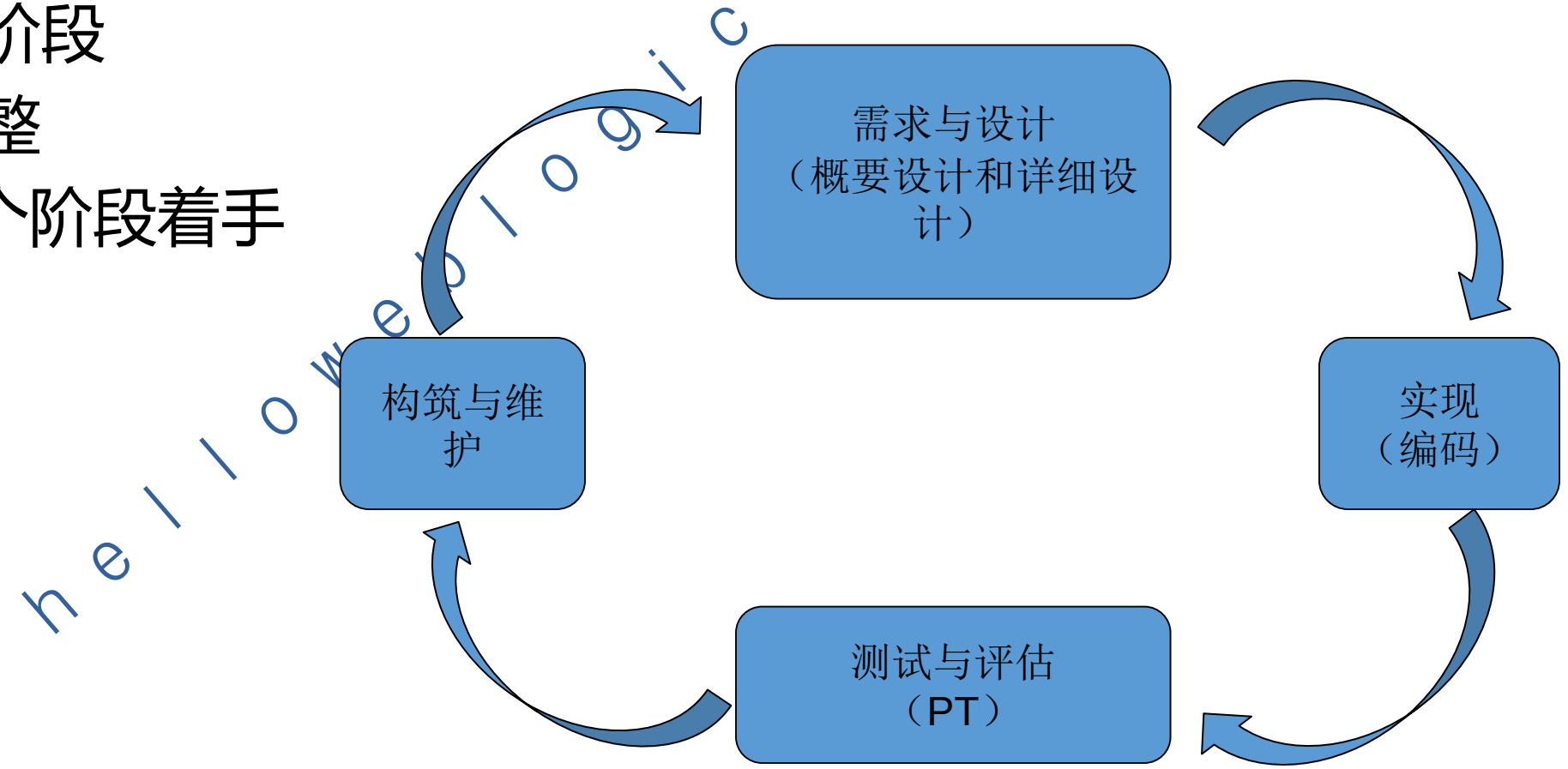
WebLogic应用性能因素

- WebLogic应用开发
- WebLogic Server本身配置
- JVM的配置
- 数据库和Web服务器
- 硬件、虚拟机、其他容器
- 其他调用系统EAI

WebLogic应用系统生命周期

- 大致四阶段
- 性能调整

  从每一个阶段着手

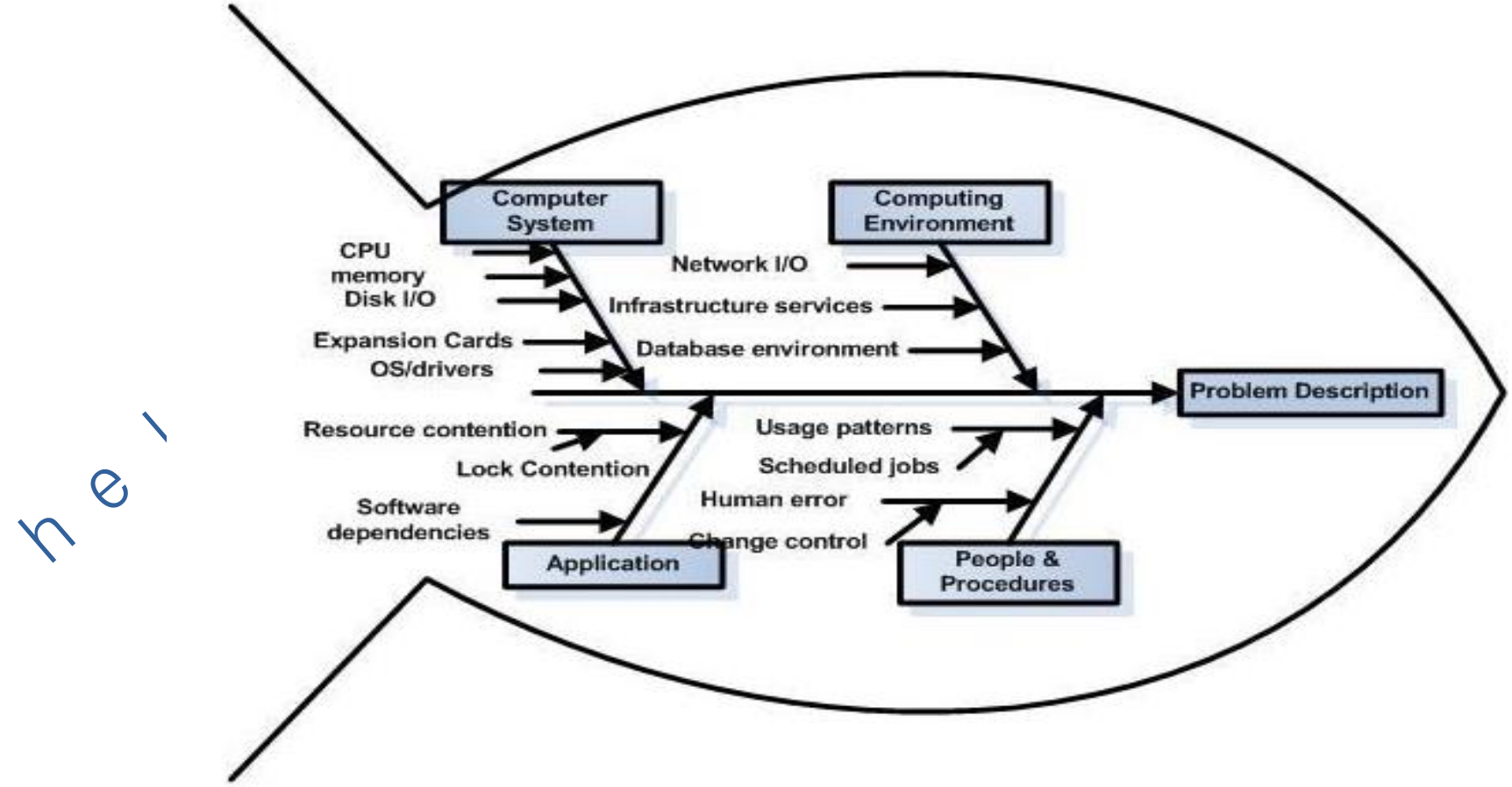


经验：提高性能的一般原则

- 设计和编码者要了解模块的资源使用、时间、接口等等。
- 尽量减少数据库的大批量操作和远程调用
- 内存操作性能大于磁盘操作（如写日志），多使用缓存技术
- 严格管理session的开销，内存的使用
- 超时判断
- 异常处理
- JDBC connection的合适使用和设置，SQL优化
- 合理使用数据软删除
- 架构上使用水平和垂直的集群扩展

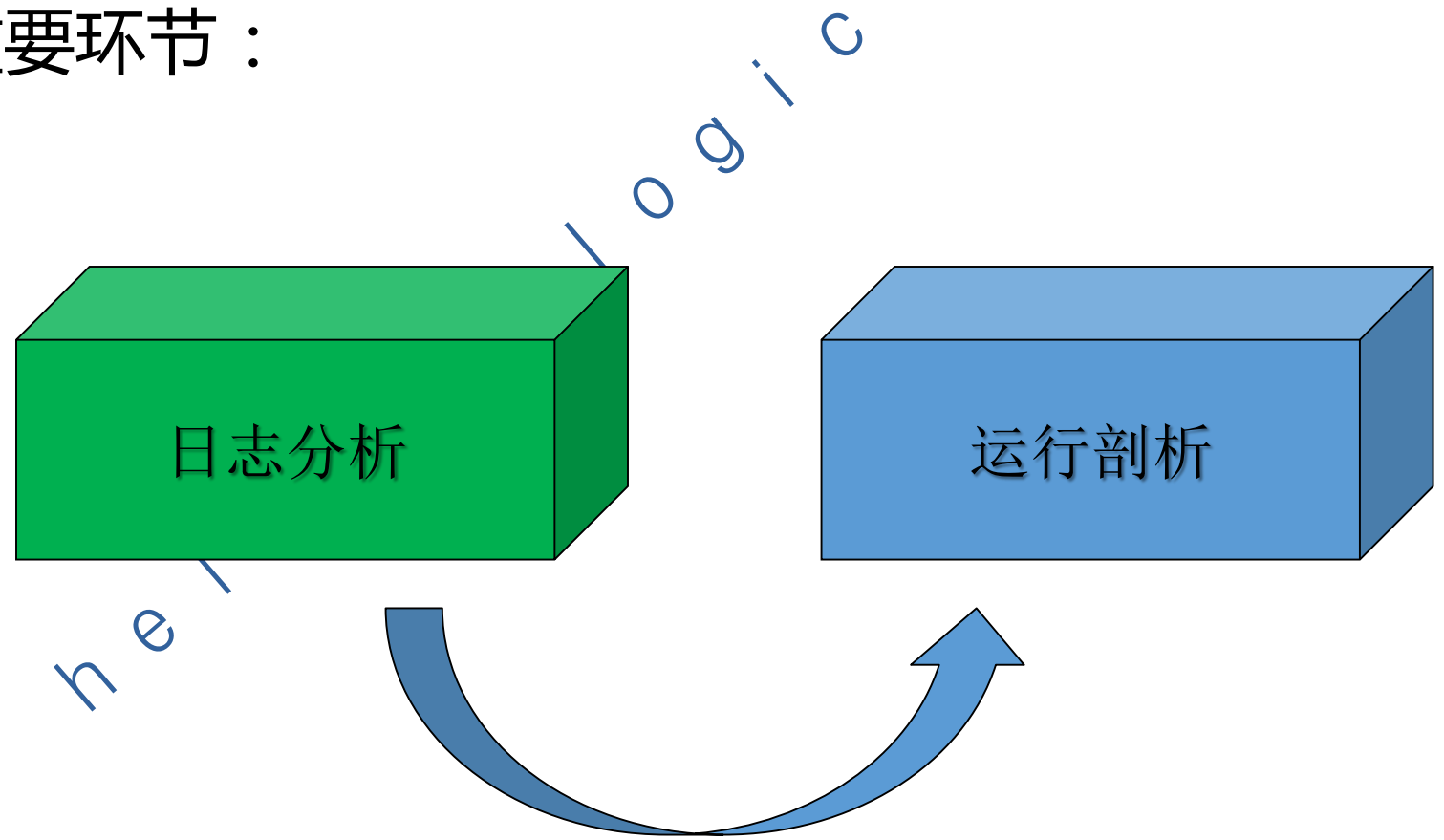
WebLogic性能问题的分析和诊断

- 信息收集-鱼骨图



WebLogic性能问题分析和诊断

- 两个重要环节：



WebLogic应用性能问题分析

- 日志分析：

① 操作系统的日志文件
② 应用的日志文件
③ Server的日志文件
④ Gc日志文件
⑤ Dump日志文件（heapdump、threaddump）
⑥ 数据库服务日志
⑦ Web服务器日志（如有，nginx、apache、haproxy等）

WebLogic应用性能问题分析

- 运行剖析（profiling)：

① Heap分析
② Thread分析
③ APM监控（oneApm,Appdynamics…)
④ 其他监控产品

案例：某客户电子商务系统优化诊断

- 主要问题：

  - 应用程序分页问题：根据 DBA 数据库检查结果以及经电子商务项目组同事确认，导致数据库资源消耗以及前端性能低下的大部分业务与应用程序分页相关。
  - OOM(OutOfMemoryError)：根据电子商务项目组同事反映，电子商务 WAS 平台运行时经常性出现 JVM 内存溢出(java.lang.OutOfMemoryError)。
  - 数据库全局死锁：根据 DBA 数据库检查结果，在数据库端经常性出现数据库全局死锁问题。目前，发生的频率为 2~3 次/天。
  - 性能低下的 SQL：根据 DBA 数据库检查结果，在数据库端存在大量的性能低下的 SQL 消耗主机及数据库资源。

案例：某客户电子商务系统优化诊断

- 优化目标：

1、 应用程序分页改造:针对应用程序分页（返回结果统计）功能引起的数据库资源消耗进行中间件层面的应用改造。针对性能问题突出业务查询，进行改造以后，在响应时间上可以减少一半或以上。

2、 故障初步诊断：主要包括针对 OOM 以及数据库全局死锁问题的初步诊断。鉴于故障分析、故障重现以及故障解决时间不可控的因素，东方龙马在工作期内将尽可能快地推进问题分析及解决，但不能保证在工作期内完全解决故障。

3、 数据库优化：主要包括 TOP SQL 优化以及针对运价查询存储过程的优化。

通过实现上述的目标，从整体上提升电子商务项目的性能以及稳定性。

案例：某客户电子商务系统优化诊断

- 数据库方面：

根据 DBA 数据库检查结果，以下 SQL 语句消耗大量的数据库资源，并且性能低下：

```
SELECT COUNT(*)
  FROM (SELECT aa.*, ROWNUM r
          FROM (SELECT n.KC_NORDERNO                orderNO,
                       n.PNR_SPNRNO                 pnrNO,
                       n.TK_TICKETNO                ticketNO,
                       r.RECEIPT_SRECEIPTNO         receiptNO,
                       r.RECEIPT_SRECEIPTSTUTS receiptStatus,
                       r.RECEIPT_STICKETFLAG        ticketFlag
                  FROM B2AORDER_PNR n, SYS_RECEIPT r, B2AKC_OrderInfo o
                 WHERE n.PNR_sReceiptNo = r.RECEIPT_SRECEIPTNO(+)
                   and o.KC_NORDERNO = n.KC_NORDERNO
                   and n.PNR_SSTATUS != '0'
                   and o.KC_sAgantOffice = 'XMN148'
                   and upper(n.KC_NORDERNO) like '%FX2011040652252O3%'
                 escape '\'
                 order by n.KC_NORDERNO desc) aa
         WHERE ROWNUM <= 100)
 WHERE r >= 1
```

在与电子商务项目组中讨论中已明确，SQL 属于分页功能的 SQL。从语句的结构来看，其主要实现的是统计当前结果集返回记录条数。

根据分页查询的实现方式，其主要与数据库进行两次交互：第一次进行 select count(*)统计，第二次\进行结果集查询。

案例：某客户电子商务系统优化诊断

- 优化建议：

① 数据库sql优化

② 查询结果集应用优化： 在已经有查询结果集的基础上，使用java.sql.ResultSet的last()以及 getRows()方法，移动查询结果集的游标，获取当前行号获得结果集数量，比使用 select cout(*)，可以减少数据库资源的消耗（如 CPU/IO 等）。该实现针对查询结果集响应时间长的查询，可以把查询结果集行数统计部分降低到毫秒级甚至更低。

案例：某客户电子商务系统优化诊断

java.sql.ResultSet last()&getRow() 样例

```
public static void testSQLSelectInRSStyle(String driver, String username,
    String password, String url, String selectSQL) {
  java.sql.Connection conn = getConnection(driver, username, password,
      url);
  if (conn != null) {
    try {
      java.sql.Statement stmt = conn.createStatement(
          java.sql.ResultSet.TYPE_SCROLL_INSENSITIVE,
          java.sql.ResultSet.CONCUR_READ_ONLY);
      java.sql.ResultSet rs = stmt.executeQuery(selectSQL);
      if (rs.last()) {
        int row = rs.getRow();
        System.out.println("ResultSet rows:" + row);
      }
      if(rs.first()){
        System.out.println("ResultSet cursor is move to first.");
      }
    } catch (SQLException e) {
      e.printStackTrace();
    } finally {
      try {
        conn.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
  }
}
```

案例：某客户电子商务系统优化诊断

评析：

经过对电子商务系统分页功能代码研究，其中，引起大量数据库资源消耗的 select count(*)统计语句，主要是由 com.iss.system.dao.impl. PageLoaderImpl.fetchData 方法导致的：在查询第一页(firstpage)，下一分组(nextgroup)，转到第 n 页功能时，会导致两次的数据库交互。对于性能问题比较突出的业务功能（查询），其两次的数据库交互，其消耗时间到少是 2*n（n 为执行一次结果集选取所需要的时间）。

因此，对于方法 com.iss.system.dao.impl. PageLoaderImpl.fetchData 的改造，在计算结果集行数的同时，把显示结果所需要的结果集同时选取出来，可以减少一次数据库交互。

案例：某客户电子商务系统优化诊断

- OOM问题

Java 进程内存溢出(java.lang.OutOfMemoryError)

1、开启 GC 日志报告：GC 日志可以用于分析应用程序对内存请求的详细情况，以及了解 JVM 对内存进行管理的详细信息，有助于确定 OOM 时，JVM Heap 的使用情况，如空闲空间，连续内存空间，应用程序请求分析空间等。

2、调整 WAS 重启机制：当前，电子商务应用配置有 Tivoli 监控软件，并在应用服务器 JVM 进程达到一定的阈值时，自动对其进行重启，该配置可能会影响 OOM 时，JVM HeapDump 文件的正常生成。

3、根据最新的 HeapDump，结合 GC 日志进行内存引用结构以及综合分析。

4、测试环境重现故障。

5、测试环境解决并尝试重现。

6、生产环境实施。

案例：某客户电子商务系统优化诊断

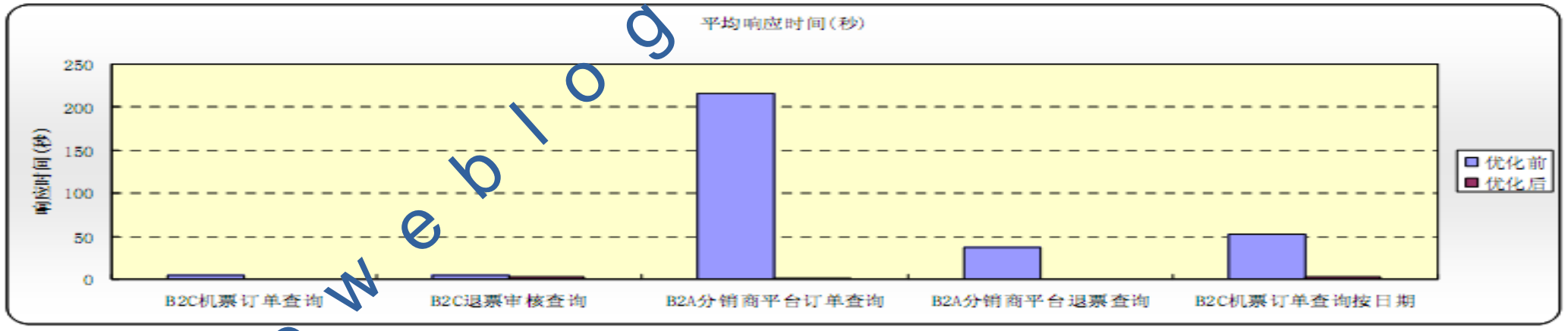
- 业务功能优化结果

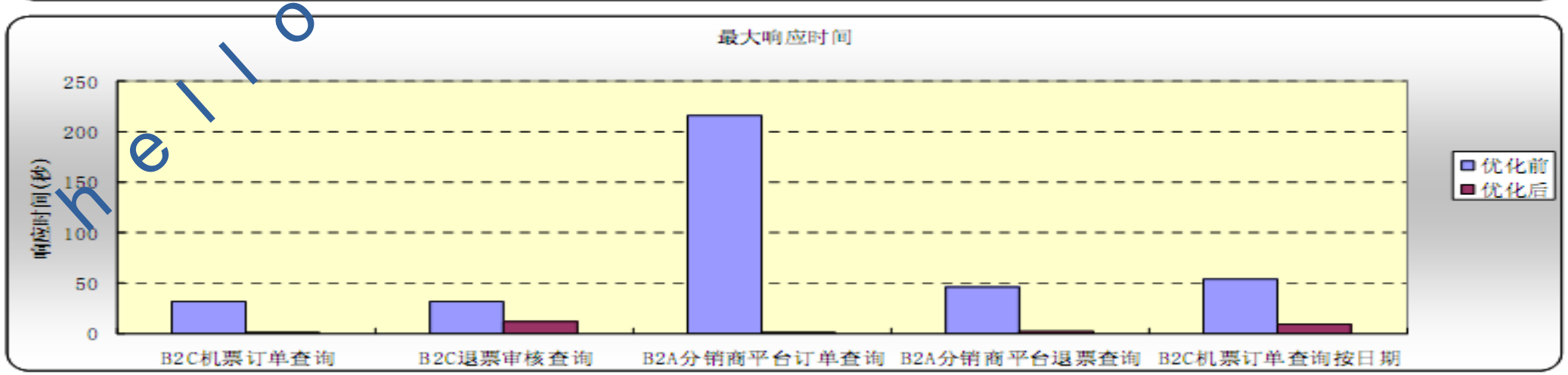
通过 LoadRunner 对以下选定的业务进行优化前后的黑盒测试结果比较进行优化效果分析：

LR指标 / 测试用例事务	Minimum		Average		Maximum		Std. Deviation		90 Percent		Pass		Fail	Stop
	优化前	优化后	优化前	优化后	优化前	优化后	优化前	优化后	优化前	优化后	优化前	优化后		
B2C机票订单查询	2.356	0.289	4.187	0.32	31.229	0.84	4.743	0.04	9.009	0.35	73	932	0	0
B2C退票审核查询	3.065	2.017	4.356	2.444	32.01	12.46	4.562	1.813	4.254	2.172	69	122	0	0
B2A分销商平台订单查询	216.422	1.089	216.422	1.313	216.422	1.817	0.004	0.078	216.422	1.418	1	204	0	1
B2A分销商平台退票查询	32.78	0.177	36.482	0.2	45.964	2.494	4.067	0.063	45.964	0.21	9	1492	0	0
B2C机票订单查询按日期	48.243	2.645	52.346	2.838	54.053	9.607	2.001	0.897	54.053	2.817	6	104	0	0

案例：某客户电子商务系统优化诊断

- 业务功能优化结果

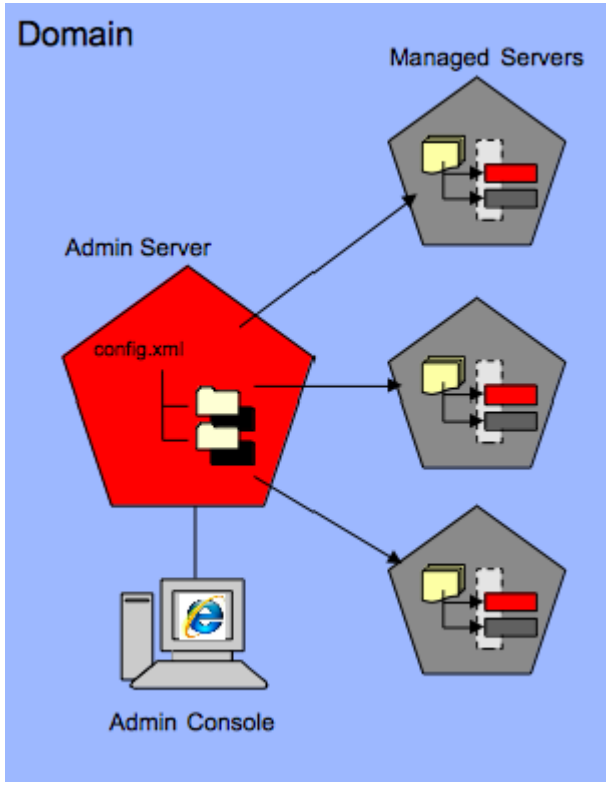




提高WebLogic应用环境性能

- WebLogic 的基础概念

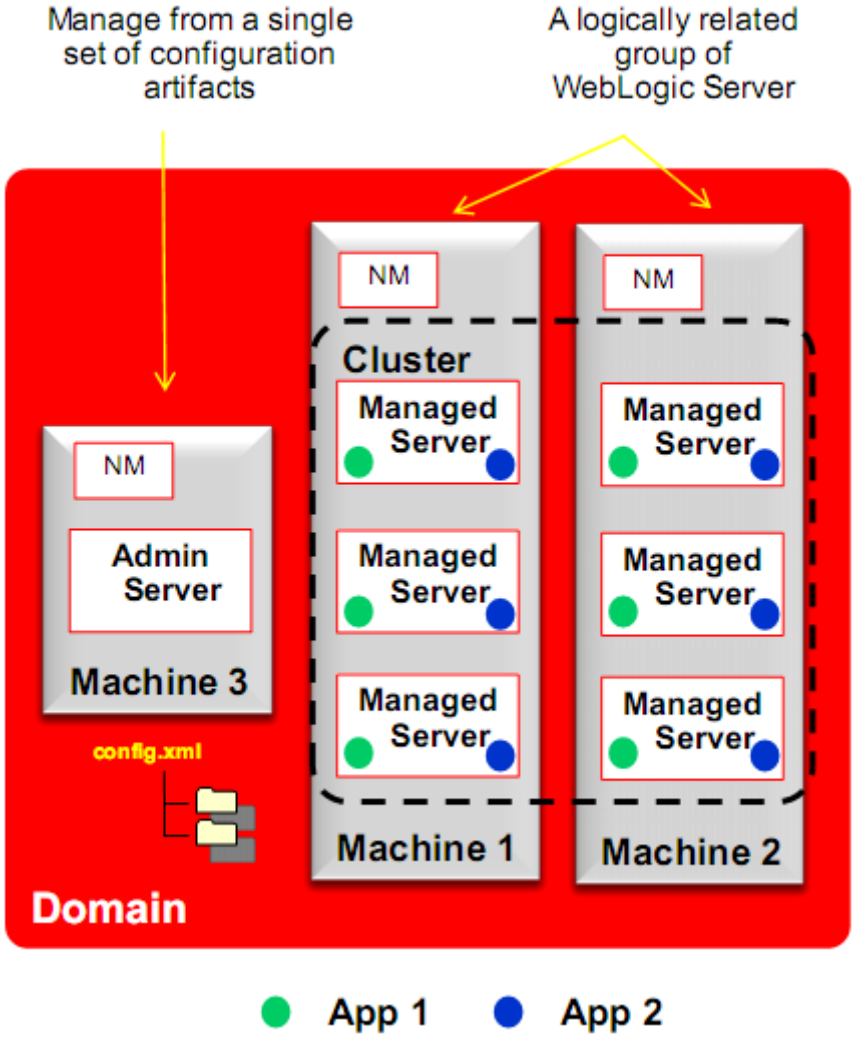
① Domain
② Server AdminServer、ManagedServer
③ Cluster
④ Noder Manager



提高WebLogic应用环境性能

- WebLogic Domain的规则

① 版本统一
② 灵活扩展
Apps/server
Servers/Cluster
Clusters/Domain



提高WebLogic应用环境性能

- 部署WebLogic Cluster-1

✓Cluder部署

-4台机器

-4个应用

-16个实例

✓单Cluster拓扑

-64个部署

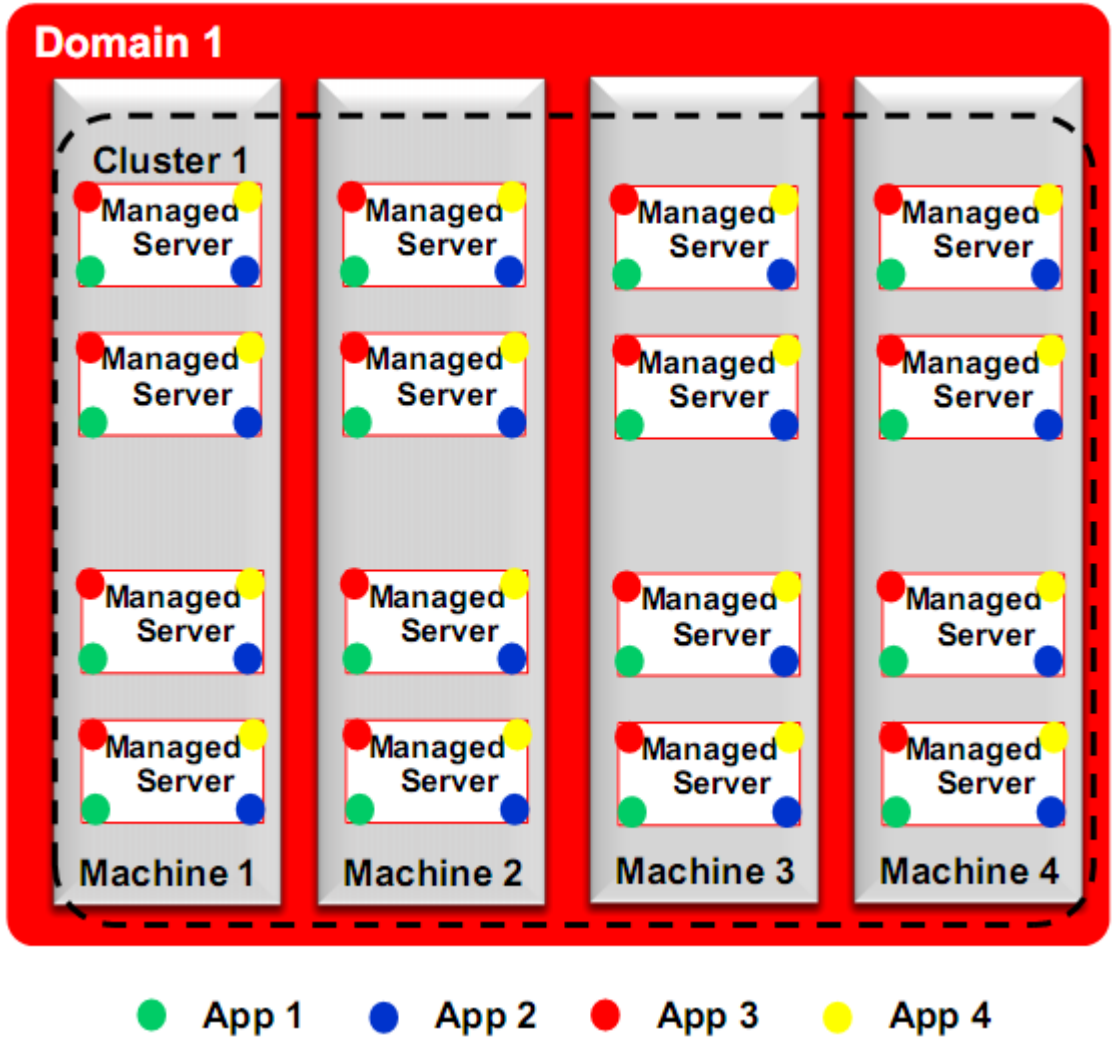
-4个Apps X 16个实例

✓扩展问题

-应用管理

-配置、维护、部署

-安全



提高WebLogic应用环境性能

- 最佳实践：WebLogic Cluster-2

✓Cluder部署

-4台机器

-4个应用

-16个实例

✓分Cluster隔离

-2个Domain

-4个Cluster

-减少部署从64减少到16

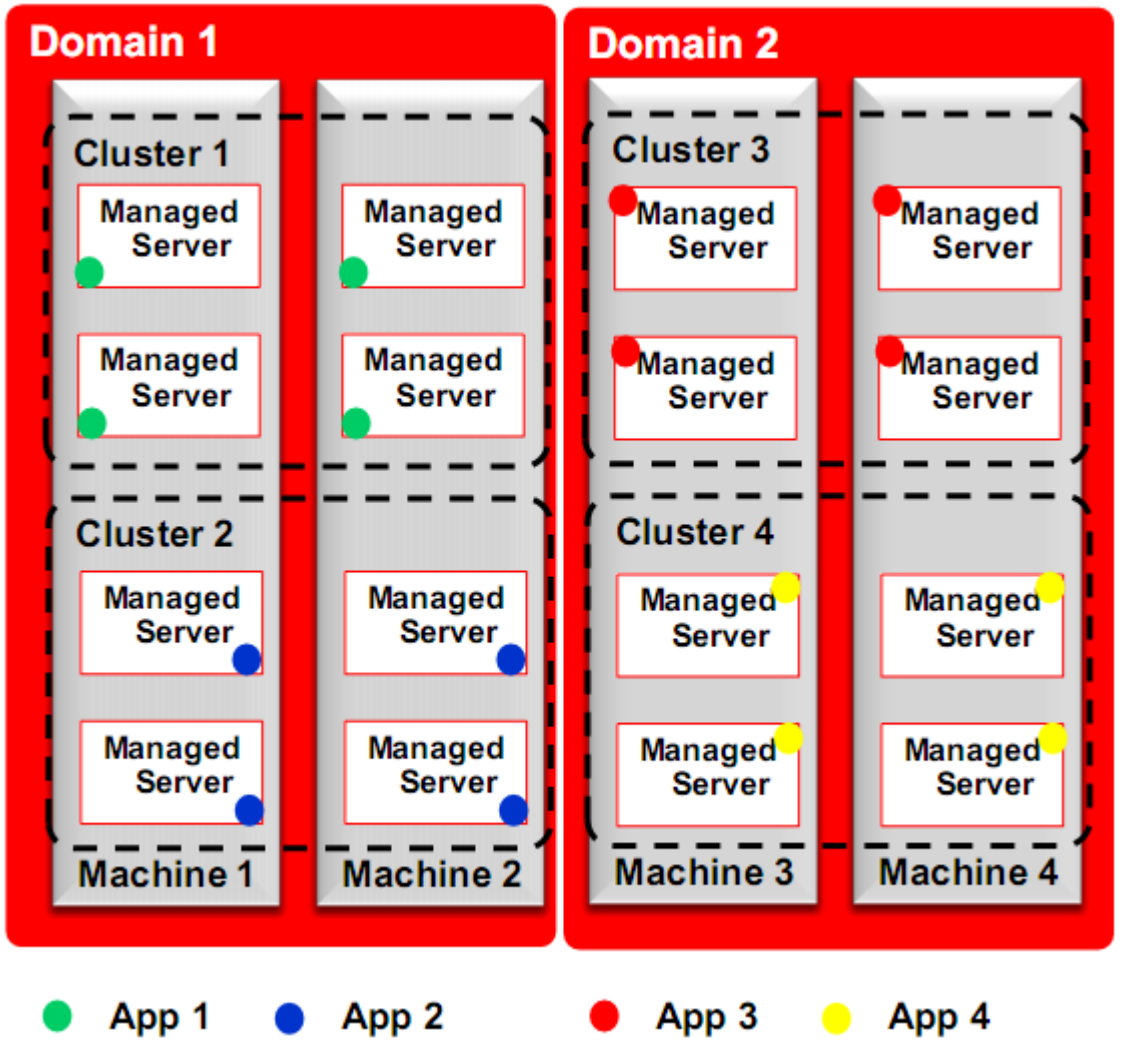
✓灵活性提高

-应用管理

-配置、维护、部署

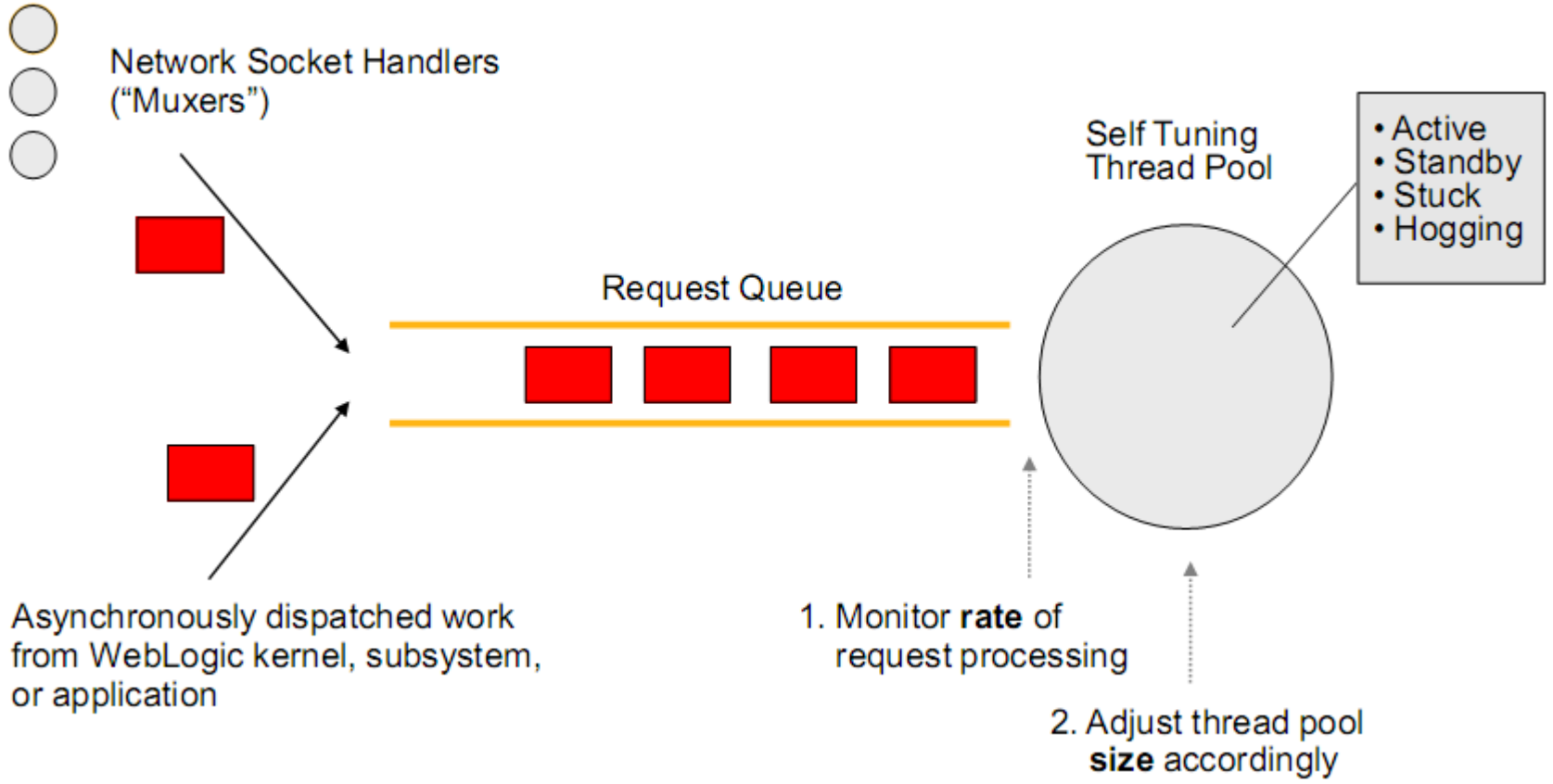
-安全

✓可扩展性提高



提高WebLogic应用环境性能

- WebLogic Work Managers Self-Tuning功能



WebLogic应用环境调优

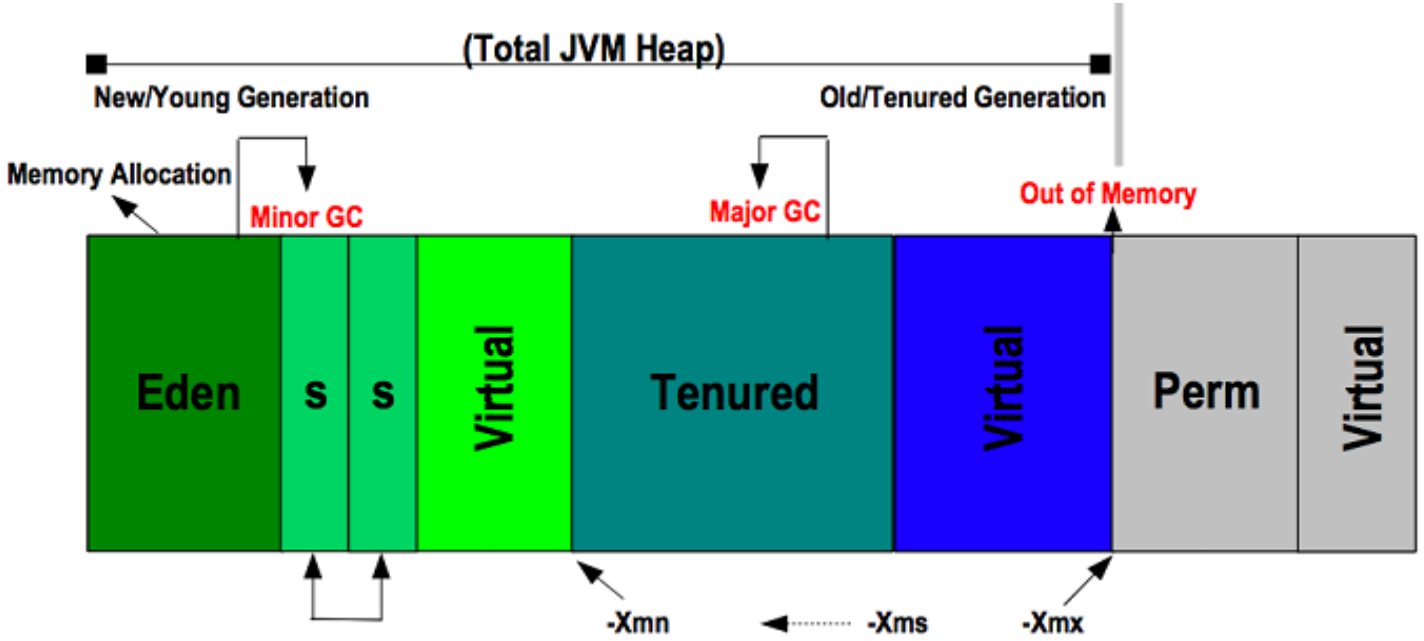
- 如下的Weblogic参数加入到startWebLogic.sh：

① -Dweblogic.SocketReaders=4
② 或者-Dweblogic.DevPollSocketReaders=4
③ 调整thread阻塞时间（默认600）-Dweblogic.StuckThreadMaxTime=1200

- 环境安装使用Oracle建议的新版JDK

WebLogic应用环境之JVM

- JVM内存管理



Young代：new()对象的实例化

2 Survivor Spaces 保存minor GC期间的对象

Old代： 持久对象

Perm代：JVM元数据

提高WebLogic应用性能-JVM参数调优

- JVM最佳实践

① 合适JVM的heap size对于weblogic非常重要

② 没有足够的Heap 空间，WLS就会OOM，宕机

③ Heap size受处理器的模式（ 32/64)的限制

④ 合理使用gc策略

⑤ 合理设置heap size提高系统性能

-- 设置-Xms == -Xmx

-- 尽量多次设置减少full gc

-- 或者设置Xms = ¼*Xmx

提高WebLogic应用性能-使用WLS动态集群

- 传统集群：

① 配置集群

② 配置Managed Server

③ 将MS 分配给集群

④ 启动MS

⑤ 将来扩展，对于每一个新增的MS

A.重复步骤 2

B.重复步骤 3

C.重复步骤 4

局限：

-- 集群成员是静态的

-- 需要动手干预才能更改配置

-- 需要动手配置MS

提高WebLogic应用性能-使用WLS动态集群

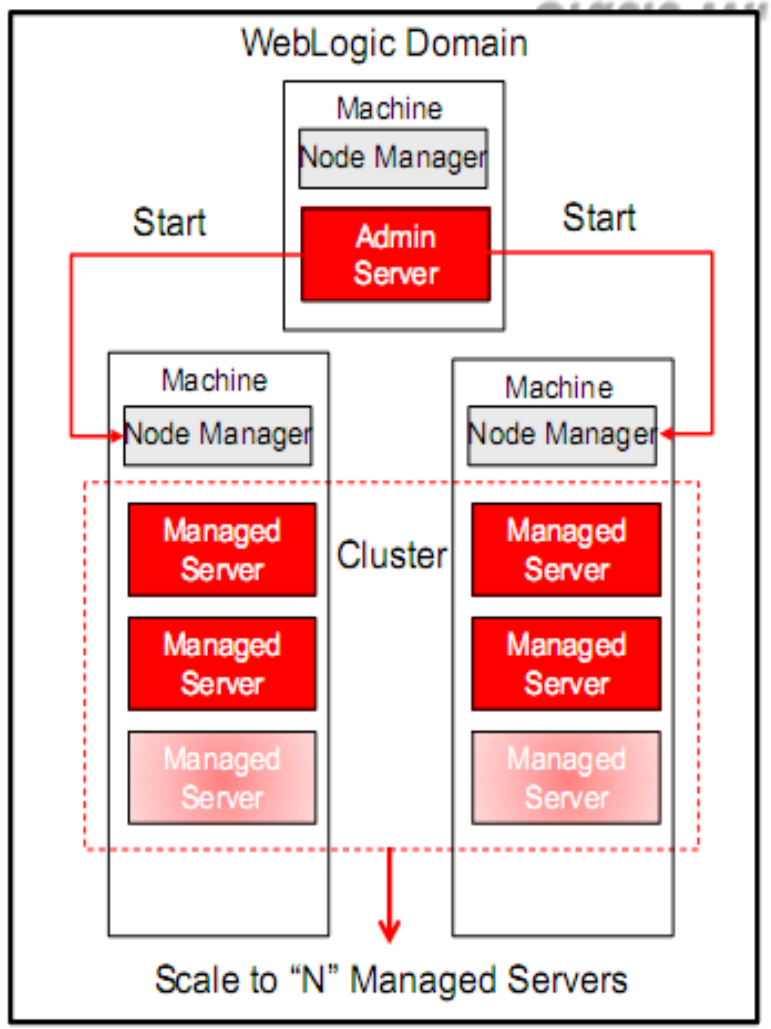
- 动态集群：

① 支持云环境的“弹性扩展”，简化集群扩展

② 采用dynamic servers和server template创建动态集群

③ 定义集群扩展时的属性：Server Name\listen port\machines等等

④ 采用标准机制启动、停止，实现集群的扩展



提高WebLogic应用性能-使用WLS动态集群

- 动态集群关键概念：
- Server Templates
  - 定义通用、非默认的属性，可以应用于不同的服务器实例
  - --一键修改，能够应用到所有的服务器实例
- Dynamic Servers
  - 所有实例不需要单独配置
  - 从Server Template获取服务器配置
- Dynamic Clusters
  - 包含Dynamic Servers 的集群

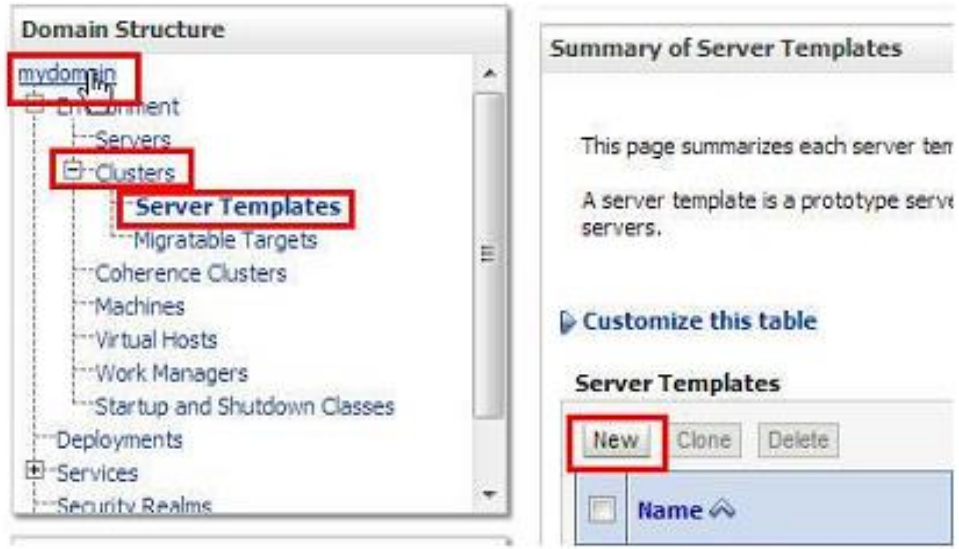
提高WebLogic应用性能-使用WLS动态集群

- 配置Server Templates：

WLST

```
1  ..
2  ..
3  serverTemplate=cmo.createServerTemplate("STemplate")
4  serverTemplate.setAcceptBacklog(2000)
5  serverTemplate.setAutoRestart(true)
6  serverTemplate.setRestartMax(10)
7  serverTemplate.setReverseDNSAllowed(true)
8  serverTemplate.setStagingMode("external_stage")
9  serverTemplate.setStartupTimeout(600)
10 soaCluster=cmo.lookupCluster("soa-cluster")
11 serverTemplate.setCluster(soaCluster)
12 ..
13 ..
```

Admin Console



提高WebLogic应用性能-使用WLS动态集群

- 动态集群：

可以通过以下方式配置

– 管理控制台

– WLST脚本

- 包含一个或者多个动态服务器
- 基于一个共享的Server Template
- 有助于在域内方便的扩展动态服务器的数量
- 预先感知高峰负载并计算服务器实例的数量
- 如果需要增加服务器，就基于Server Template 创建新的服务器
- 以下属性是配置的关键参数

– Server Name, Listen Ports, Machines, Network Access Point

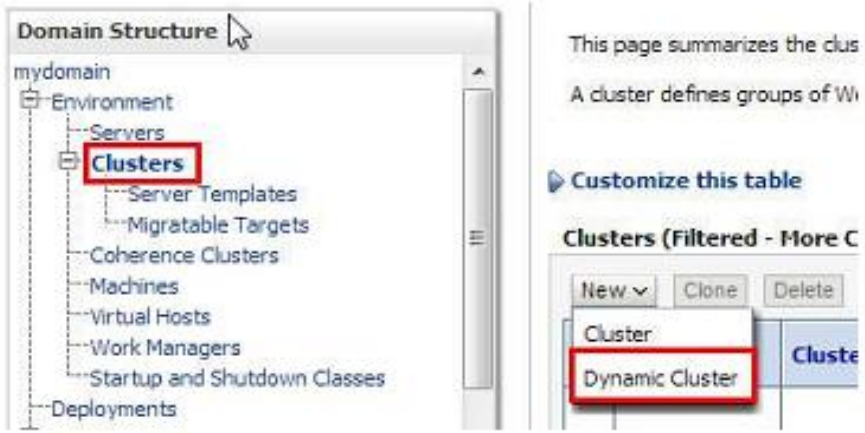
提高WebLogic应用性能-使用WLS动态集群

- 配置动态集群：

WLST

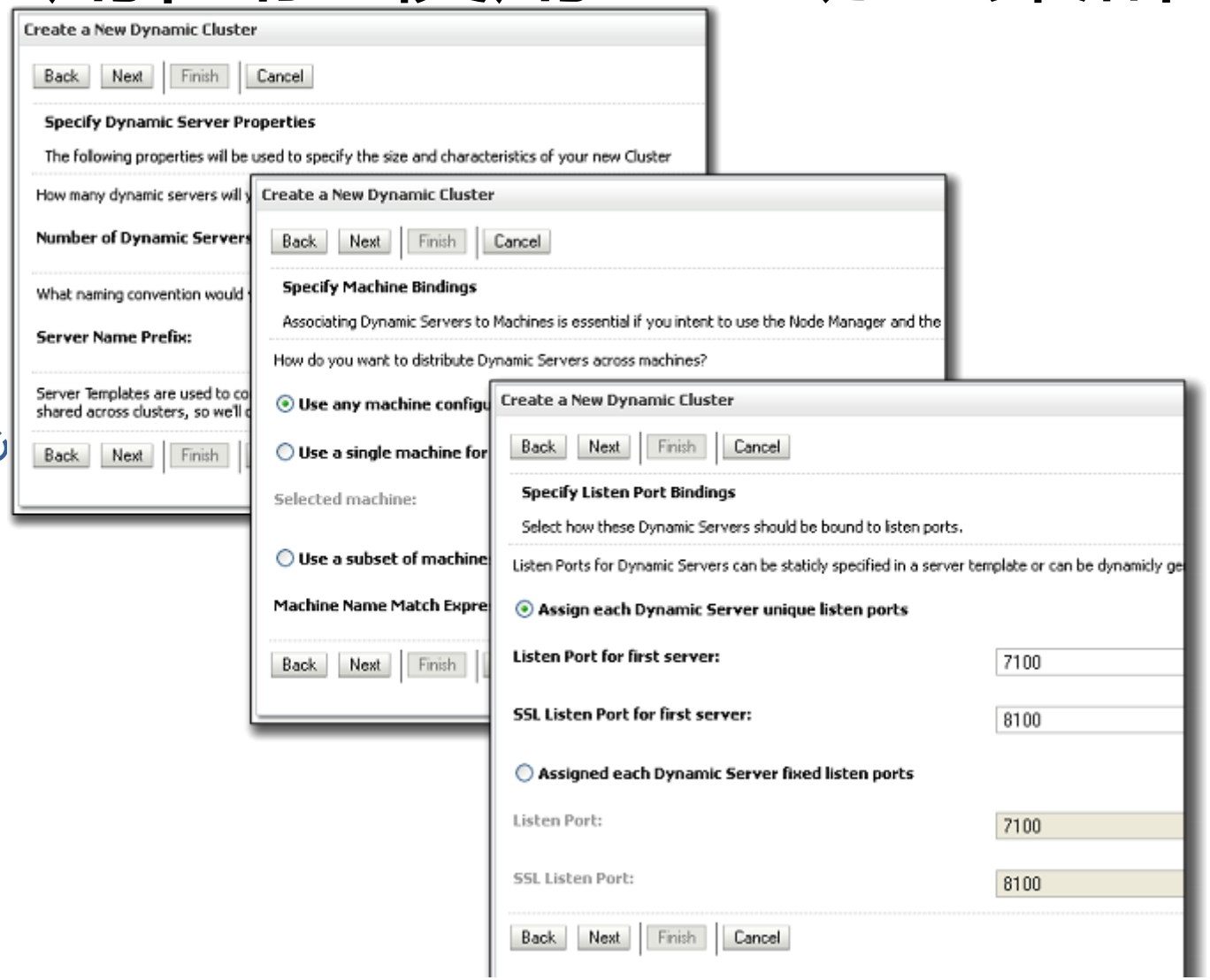
```
..
..
# create the server template for the dynamic servers and
# set the attributes for the dynamic servers. Setting
# the cluster is not required.
#
dynamicServerTemplate=cmo.createServerTemplate("dc-server-template")
dynamicServerTemplate.setAcceptBacklog(2000)
dynamicServerTemplate.setAutoRestart(true)
dynamicServerTemplate.setRestartMax(10)
dynamicServerTemplate.setStartupTimeout(600)
#
create the dynamic cluster and set the dynamic servers
#
dynCluster=cmo.createCluster("dynamic-cluster")
dynServers=dynCluster.getDynamicServers()
dynServers.setMaximumDynamicServerCount(10)
dynServers.setServerTemplate(dynamicServerTemplate)
#
# dynamic server names will be dynamic-server-1,
# dynamic-server-2, ..., dynamic-server-10
#
dynServers.setServerNamePrefix("dynamic-server-")
..
..
```

Admin Console



提高WebLogic应用性能-使用WLS动态集群

- 动态集群配置向导：
- 交互式创建方式
- 简化动态集群创建
- 导引您完成整个过程
- 最后生成:
  - 含动态服务器的集群
  - Server Template
  - Machine mappings
  - Listen address and port mappings



提高WebLogic应用性能-使用WLS动态集群

- 扩展动态集群：

① 创建Server Template

② 创建Dynamic Cluster

③ 基于Server Template 设置DynamicServers

④ 启动Dynamic Server

⑤ 重复第四步，扩展集群中的服务器

⑥ 所有的动态服务器默认为动态集群的一部分

- 动态集群好处：

① – 集群成员是动态的

② – 启动动态集群可以通过WLST / Node Manager

③ – Oracle 推荐通过管理控制台停止动态集群

④ – 可用脚本编程式启动、停止动态服务器来改变集群成员

⑤ – 配置动态服务器很简单，根据相关的Server Template读取配置

⑥ – 支持云环境

提高WebLogic应用性能-使用WLS动态集群

```
<server-template>
    <name>dctemplate</name>
    <accept-backlog>2000</accept-backlog>
    <auto-restart>true</auto-restart>
    <restart-max>10</restart-max>
    <startup-timeout>600</startup-timeout>
</server-template>
<cluster>
    <name>dynamic-cluster</name>
    <dynamic-servers>
      <server-template>dctemplate</server-template>
      <maximum-dynamic-server-count>8<maximum-dynamic-server-count>
      <calculated-machine-names>true</calculated-machine-names>
      <machine-name-match-expression>dyn-machine * </machine-name-match-expression>
      <server-name-prefix> S erver - </server-name-prefix>
    </dynamic-servers>
</cluster>
.....
```

提高WebLogic应用性能-操作系统调优

- Windows OS：

① TcpTimedWaitDelay
② MaxUserPort
③ DynamicBacklog
④ KeepAliveInternal
⑤ TcpMaxDataRetranmission
⑥ JVM内存大页支持

- AIX OS：

① TCP_TIMEWAIT
② TCP_KEEPIDLE
③ TCP_KEEPINTVL
④ TCP_KEEPINIT
⑤ JVM内存大页支持
⑥ ulimit

提高WebLogic应用性能-web服务器调优

- 对于Web服务器通常需要关注下列配置：

① 进程或线程的监听器

② 静态图片的分目录保存

③ 日志

④ 合理使用socket


```
<IfModule worker.c>
 ThreadLimit 25
 ServerLimit 64
 StartServers 2
 MaxCliemts 600
 MinSpareThreads 25
 ThreadsPerCild 25
 MaxRequesPerChinld 0
</IfModule>
```

提高WebLogic应用性能-JDBC和数据库调优

- 对于JDBC：

① 连接池的大小参数
② 增长和扩展
③ 语句的缓存
④ 连接超时
⑤ 连接池泄露分析
⑥ 使用高速数据缓存，如memcache, conhenrence等等

- 对于数据库：

① 略

提高WebLogic应用性能-高速缓存

- 使用一些数据缓存的软件
- Memcache\Coherence\内存数据库

本次分享总结

- 什么是性能问题
- WebLogic应用系统的四阶段
- 提高WebLogic应用性能的一般原则
- 怎么分析和诊断WebLogic性能问题，（案例应用代码方面）
- 实践：部署WebLogic集群和动态集群
- 实践：JVM参数调优
- 实践：操作系统调优
- 实践：web服务器调优
- 实践：JDBC参数调优

THANKS FOR WATCHING