Glassfish 应用服务器产品对比白皮书

内容目录

Glassfish 简介

Sun Glassfish Enterprise Server 是业界最受关注的应用服务器产品。Sun 公司于 2005 年 6 月启动了 GlassFish 项目——开发一个与 Java Platform Enterprise Edition 5 (Java EE 5) 兼容的应用服务器产品，并向 Java.NET 社区开放源代码。2006 年 5 月，Glassfish V1 作为 Java EE 5 的参考实现，和 Java EE 5 规范同时发布。Glassfish 是第一个开源的、与 Java EE 5 兼容的应用服务器。2007 年 9 月，GlassFish 社区发布 GlassFish v2，其中包括 GlassFish v1 的所有功能，并添加了其他功能，使得应用服务器能够接受重型生产环境的挑战。Sun Glassfish Enterprise Server 是 Sun 支持的商用应用服务器版本，他和 Glassfish 社区版本具有相同的源代码，SUN 公司对该商业版本提供全面的支持和保障。

应用服务器市场分析

目前的应用服务器领域，基本上可以分为两大阵营：商用和开源。开源应用服务器产品有 Glassfish, Jboss, Geronimo, Tomcat, Resin 等，这里只有 Glassfish 和 Jboss 5 是完全 Java EE 5 兼容的，Tomcat 和 Resin 只提供了 Web Containner，严格意义上不能是一个完整的应用服务器产品。很多开源应用服务器都有对应的商用版本，例如 Glassfish 对应的 Sun Glassfish Enterprise Server, Jboss 对应的 JBoss Enterprise Application Platform，Resin 对应有 Resin Application Server Enterprise Edition. 这里只有 Glassfish 的开源版本和商用版本具有相同的源代码，其他开源应用服务器的商用版本，都具有不同的 code base，而且商用版本是不开源的。在商用应用服务器市场，目前的主导者还是 IBM Websphere 和 Oracle Weblogic. 他们仍然占有企业领域的大部分市场份额。开源应用服务器也做得越来越好，比如 Glassfish，本身就是 Sun 的商用应用服务器产品开源到 Java.Net 社区形成的 Glassfish 开源项目，在企业级特性，比如高可用性、可伸缩性、集群、综合监控、SOA、控制台易用性等方面具有不输于商用应用服务器的能力，能够接受重型生产环境的挑战，又具有开源的优势。下面篇章我们将着重对开源的 Glassfish 应用服务器和商用的 Weblogic 应用服务器做一些对比。

Glassfish 和 Weblogic 发行版本比较

Glassfish 所有版本都是 Java EE 5 兼容的，Weblogic 从版本 10 开始兼容 Java EE 5 规范。因而我们的比较将针对 Glassfish V2.1 和 Weblogic 10 这两个在相同规范下的产品。
首先从发行版本上讲，Glassfish 只有开源和商用两个版本，而且这两个版本具有完全相同的源代码基础，功能、性能上没有任何不同。因而可以说 Glassfish 商用版本和开源版本唯一的不同，就在于商业服务，用户可以根据需要购买不同的服务等级。

Weblogic 应用服务器目前有 2 个版本：Oracle WebLogic Server 10g Standard Edition 和 Oracle WebLogic Server 10g Enterprise Edition。 其中 Standard Edition 实现了基本的 Java EE 5 规范的内容，但不能做高可靠性集群。Enterprise Edition 中加入了对集群的支持、多域管理和诊断工具。

功能对比

下面我们将针对 Weblogic 的两个发行版本和 Glassfish 进行功能比较

优势	描述	特性	WLS Standard	WLS Enterprise	Glassfish
通过 Java EE 5 兼容性认证	实现了 Java EE 5 规范	Servlets (2.5, 2.4, 2.3, 2.2)	●	●	●
		JSP 2.1	●	●	●
		JSF	●	●	●
		JSTL	●	●	●
		JDBC 3.0	●	●	●
		JNDI	●	●	●
		JMX	●	●	●
		JTA	●	●	●
		JCA 1.5	●	●	●
		EJB 3.0	●	●	●
		Java Persistence API	●	●	●
		JMS 1.1	●	●	●
		JAAS 1.2	●	●	●
		JDO 2.0	●	●	●
提高开发效率	应用开发的工具支持和框架支持	Ant-based build tools	●	●	●
		Netbeans 6.5	●	●	●
		Eclipse 3.3.2	●	●	●
		Oracle TopLink Essential	●	●	●
		Hot deployment	●	●	●
		Hot modification	●	●	●
		Optional service startup	●	●	●
		Lightweight installers	●	●	●

		Annotations and dependency injection	●	●	●
		Spring Framework support	●	●	●
		Struts Framework support	●	●	●
投入生产可靠性	确保应用可以访问，可靠更新数据。	Application failover to backup database via JDBC connection pooling	●	●	●
		Multiple data sources to support connections to Oracle RAC or other database cluster with load balancing, failover, and XA support	●	●	●
		Application load balancing via JDBC connection pooling	●	●	●
企业强度的高可用性和扩展性	集群能力，确保错误恢复、状态复制、负载均衡	Session replication and failover		●	●
		Whole server migration		●	●
		Asynchronous HTTP session replication reduces latency in HTTP client response		●	●
		Highly available singleton service management		●	●
		In-memory replication of EJB state		●	●

		In-memory replication of servlet session state		●	●
		Stateful session EJB failover		●	●
		Clusterwide JNDI naming service		●	●
		Automatic migration of Transaction Recovery Service (TRS)		●	●
高性能	可以让应用运行更快速的特性	Grizzly(NIO)			●
		Web caching	●	●	
		Thread pooling, connection pooling, multipools	●	●	●
		Oracle JRockit JVM	●	●	
管理特性	易用的应用程序管理特性，综合监控能力	Web-based administration console, providing monitoring and configuration of instances, resources, and applications	●	●	●
		Wizard-based domain and cluster configuration	●	●	
		CLI-based domain and cluster configuration			●
		Programmatic management via JMX	●	●	●
		SNMP 3.0 support	●	●	●
		Monitor & diagnostics		●	●
服务器自调整	简化 Glassfish 性能配置过程	prioritize work based on run-time	●	●	●

		performance and throughput			
		allows work to fail over to another server in a clustered environment when system capacity is reached		●	●
安全	访问控制，安全认证服务	Pluggable security modules for authentication, authorization, auditing, and PKI management			
		Built-in LDAP-based data store for all profile and entitlement data	●	●	●
		Graphical security policy administration editor	●	●	●
		Automatic conversion of existing access controls list (ACL)	●	●	●
消息系统	JMX 消息	Point-to-point, publish and subscribe, durable subscribers, XA compliant, XML messaging, unit of order, unit of work	●	●	●
		Asynchronous data processing with message-driven beans	●	●	●
		Plug and play with third-party messaging providers	●	●	●
		JMS functionality		●	●

		distributed across clusters			
		Highly-available connection factories and destinations		●	●
		Automatic migration of messaging/JMS		●	●
		Reliable message with store and forward		●	●
Web Service	构建基于 Web 服务的分布式应用	Support for SOAP, XML, JAX-RPC, UDDI, and WSDL	●	●	●
		Interoperable with services tool kit	●	●	●
		Web services metadata for the Java platform (JWS) 2.0, 1.0	●	●	●
		Java EE 5 conformant Web services, including EJB-based Web services, SOAP/JMS	●	●	●
		JAX-WS 2.1 and JAXB 2.1, 2.0	●	●	●
		JAX-R 1.0	●	●	●
		MTOM: high performance transport of binary XML data as MIME attachment	●	●	●
		Advanced Web services, such as stateful conversations and buffered services	●	●	●
		SOAP 1.1, 1.2	●	●	●
		SOAP with attachments API for Java (SAAJ)	●	●	●

		WSDL 1.1	●	●	●
		UDDI 2.0	●	●	●
		Java EE Enterprise Web services 1.2, 1.1	●	●	●
		JAX-RPC 1.1	●	●	●
		WS-SecureConversation 1.3	●	●	●
		WS-Security 1.1, 1.0	●	●	●
		WS-SecurityPolicy 1.2	●	●	●
		WS-Policy 1,2, 1.5	●	●	●
		WS-PolicyAttachment 1.0	●	●	●
		WS-Addressing 1.0, 2004/08 member submission	●	●	●
		WS ReliableMessaging	●	●	●
		WS-Trust	●	●	●
		SAML 2.0	●	●	●
		SAML Token Profile 1.1	●	●	●
		Highly-available Web services	●	●	●
分布式事务管理	使得用户数据在多个系统组件之间保持一致	Two-phase commit protocol support	●	●	●
		Advanced transaction services including fault tolerance and transaction monitoring	●	●	●

系统集成	与主流软硬件的兼容性和互操作性	J2EE Connector Architecture (1.5)	●	●	●
		Microsoft COM+ Interoperability	●	●	●
		CORBA interoperability through IIOP	●	●	●
		Pluggable messaging infrastructure allows integration with other JMS messaging solutions	●	●	●
		Pluggable security allows integration with other security solutions	●	●	●
嵌入 Web 服务器支持		HTTP, servlets, JSP, CGI	●	●	●
		Plug-ins for Apache, Sun Java System Web Server	●	●	●
		Compatible with third party hardware load balancers such as F5 BIG IP	●	●	●
		Virtual hosting	●	●	●
多种平台支持	支持多种操作系统	Certified on most major hardware and operating systems including Unix, Linux, and Windows	●	●	●

操作系统平台支持

系统需求

操作系统	AIX HP-UX Linux Solaris Windows	Solaris Windows Linux MacOS AIX
数据库	Oracle IBM DB2 Microsoft SQL Server MySQL Sybase	Oracle IBM DB2 Microsoft SQL Server MySQL Sybase JavaDB
Java	Java Platform, Standard Edition 6 (clients only) Java Platform, Enterprise Edition 5	Java Platform, Standard Edition 6 or Java Platform, Enterprise Edition 5

应用服务器的性能

Glassfish 和 Weblogic 都具有业界领先的性能优势。对应用服务器性能的考量，目前大家公认的评判标准是 SPECjAppServer2004，它是行业标准的基准测试规范，用于度量 J2EE 硬件和软件平台的性能和可伸缩性。它是由 SPEC 的 Java 小组委员会（包括 BEA、Borland、Darmstadt University of Technology、Hewlett-Packard、IBM、Intel、Oracle、Pramati、Sun Microsystems 和 Sybase）开发出来的测试标准。它实现了新的增强的 workload，涉及到所有主要的 Java EE 平台服务，包括：

* Web 容器，包括 servlet 和 JavaServer 页面
* EJB 容器
* EJB 2.0 容器托管持久性
* JMS 和消息驱动 bean
* 事务管理

* 数据库连通性

根据厂商发布的测试数据，我们看到 Glassfish 在 SPECjAppServer2004 基准测试中获得 883.66 JOBS，而 Weblogic 获得 801.70 JOBS（该测试发布的硬件平台是 SUN T2000 服务器）。从 SPECjAppServer2004 数据中我们看到，Glassfish 和 Weblogic 在性能上都处于领先的地位，而 Glassfish 还要更好一些。

附加特性

服务器端脚本支持

Glassfish 支持很多种脚本语言，是 Web2.0 的理想选择。比如我们可以在 Glassfish 上部署 php，JRuby on Rails， Groovy on Grails， Jython and Django 等很多脚本语言和动态语言应用程序。

监控能力

Glassfish 的 CallFlow Monitoring 使得开发人员和系统管理员可以监控已部署的应用程序的运行情况。

CallFlow 的监控是通过 AMX Mbean 实现的。通过 AMX 可以访问到 CallFlow 的诸多特性，包括：控制 CallFlow 监控的开/关、获取 CallFlow 请求和响应信息。

应用程序的执行都是由一系列的方法调用来完成的。这一系列方法的调用是否按照开发时设想的次序在执行、每个方法调用时间的长短，这些都是开发和管理人员所关心的。

CallFlow 就是用来帮助解决这些问题的。CallFlow 可以看作是应用服务器内置的性能探查器，它收集应用程序运行时的各类信息，包括各类容器中消耗的时间，被调用的方法及其响应堆栈，关联的应用程序名和模块名，异常的出处等。可以利用 CallFlow 收集的这些信息来进行性能调优和程序调试。此外，除了收集数据，CallFlow 还有数据查看和过滤的功能。

.net 互操作性

Glassfish 和 Weblogic 都提供了 Java 和.NET 互操作的能力。 Java 虚拟机 JVM 和.NET 通用语言运行时 CLR 都提供了程序运行所需的功能服务，其中包括内存管理、线程管理、代码编译（或 Java 特有的即时编译 JIT）等等。由于这些特性的存在，在一个操作系统中，如果程序同时运行在 JVM 和 CLR 两种环境之上，由于任何一个进程都可以加载与之对应的任何共享类库，这使得相应的操作将变得非常繁琐。

Glassfish 社区的 Tango 项目，提供了通过 webservice 和.NET 互操作的框架。通过 Netbeans IDE，可以很方便的实现 glassfish 和.Net 的互操作。

Oracle 与 BEA 也已经加入到 Sun 的 Web 服务规范实现栈(Web Services stack)Tango 项目中来，他们不会再分开而各自为政。Glassfish 和 Weblogic 中，对.Net 互操作的实现，都是基于 java.net 的 Tango 项目。在 WebService Stack 实现和.Net 的互操作，对任何应用服务器来说，都是很重要的。

控制台

Glassfish 和 Weblogic，都提供企业级强大的 Web 控制台，他们都提供了图形化的管理界面和监控界面。此外，Glassfish 还提供了 CLI(Command Line)方式的系统管理模式，给系统管理员以最大的自由度。Glassfish 和 Weblogic 都提供 Ant 脚本的方式进行管理，方便与用户的 Solution 集成。此外，Glassfish 还提供了 API 的方式进行管理，例如，以下代码可以在程序里创建 Glassfish 运行时环境，并执行 Servlet：

```
public void testServlet() throws Exception {
        int port = 9999;
        GlassFish glassfish = new GlassFish(port);
        URL url = new URL( “http://localhost:” + port + “/test” +
 “/SimpleServlet” );
        BufferedReader br = new BufferedReader(
                new InputStreamReader(url.openConnection().getInputStream()));
        assertEquals( “Wow, I ‘m embedded!”, br.readLine());
        glassfish.stop() ;
}
```

高可用性

集群是企业应用的应用服务器选择中需要考虑的因素。通过集群，Web 应用可以实现负载均衡和错误恢复。Glassfish 和 Weblogic 配置 Cluster 都非常简单，这是企业级应用服务器的重要特征。Glassfish 集群采用 JXTA 协议的 p2p 网络计算平台实现，极大简化了配置过程。通过 Glassfish 集群，可以实现 http session 状态复制，Stateful EJB 的 Session 状态复制，单点登录的状态复制，容器的状态复制等等，从而确保了 Web 应用不会因为单点故障而导致用户数据丢失。Glassfish 高可用性的配置过程，可以参见以下在线视频：
http://dl.getdropbox.com/u/150028/sun_tech_day/LiveDemo_GF_Cluster.avi

附录：从 Weblogic 迁移到 Glassfish 指南

Overview

本文将详细介绍把 Weblogic 9.2 的 SimpleTag Sample 移植到 Glassfish v2 (Sun Java System Application Server 9.1 的详细过程。通过这个实际例子，我们能够了解到 Weblogic 9.2 和 Glassfish v2 在部署方式，资源注入，所支持的数据库等方面的差别；从而大致了解应用服务器移植过程中面对的挑战以及应对的方法。

Software packages and Tools

本次实验运行于 Windows XP 和 OpenSolaris，所用到的软件包和工具是：

- Weblogic 9.2 for Windows 32bit (http://download2.bea.com/pub/platform/92/server920_win32.exe)
- Netbeans 6.1 + Glassfish v2 + MySQL 5.0 Community Server bundle (http://download.netbeans.org/netbeans/6.1/mysql_bundle/netbeans-6.1-mysql-solaris-x86.sh)
- JDK (Java Development Kit) 6 Update 12 (http://java.sun.com/javase/downloads/index.jsp)

注：如果实验时间比较紧张(0.5 hour)的话，请在实验之前把上述软件安装好。

Simple Tag Sample Application

SimpleTag Sample 是随着 Weblogic 9.2 一起发布的例子中的一个，它主要用来介绍怎么在 JSP 页面当中使用 Simple Tag API 来对一个数据库做查询。我们的移植过程主要分以下几个部分：

- 数据库，从 PointBase -> MySQL
- Java EE 5 资源注入的使用，使数据库访问的代码更通用
- 虚拟目录映射(Virtual directory mapping)在 Glassfish 里的实现

Weblogic SimpleTag 的所有源代码在 $WLS_INSTALL/weblogic92/samples/server/examples/src/examples/webapp/jsp/tags/simple 里面可以找到。其中$WLS_INSTALL 是 Weblogic 的安装目录，缺省是 C:\bea

Step 1: (Windows) WLS Split Directory -> GF Exploded Directory

Weblogic 使用一种所谓 Split Directory 的开发方法，简单来说就是将源代码(jsp, java, xml 等)和编译产生的 class 文件分别存放于不同的目录。关于 Split Directory 的详细介绍，请参照 http://e-docs.bea.com/wls/docs91/programming/splitcreate.html

为了部署这样的应用，我们需要对应用重新打包。Weblogic sample 提供的编译文件 build.xml 里面有一个目标 package.exploded.ear 可以帮我们做这个工作：(这是本次实验唯一需要在 Windows 下面进行的操作，因为我们必须使用 Weblogic 自带的 ant 来编译 package.exploded.ear 目标。)

注：如果实验时间比较紧张(0.5 hour)的话，该步骤由 speaker 在 windows 上演示，audience 直接使用生成后的包(jspSimpleTagEar.zip)在 OpenSolaris 上面进行后续的步骤。

- cd c:\bea\weblogic92\samples\server\examples\src\examples\webapp\jsp\tags\simple
- c:\bea\weblogic92\common\bin\commEnv.cmd
- ant build
- ant package.exploded.ear

上面的编译命令会产生Glassfish能够支持的所谓exploded format的目录结构:

```
WebLogic:  Exploded  EAR
jspSimpleTagEar
      META-INF
      ..  application.xml
      ..  weblogic-application.xml
      jspSimpleTagWar
      ..  ExamplesFooter.jsp
      ..  ExamplesHeader.jsp
      ..  SimpleTag.jsp
      ..  wls_examples.css
      ++  WEB-INF
            ..  web.xml
            ..  weblogic.xml
            ++  classes/examples/webapp/jsp/tags/simple
                  ..  ExecuteSql.class
                  ..  IteratorTag.class
            ++  tlds
                  ..  simpleTag.tld
```

我们要对它作简单的修改：

- 删除weblogic-application.xml，它是一个空文件
- 将weblogic.xml替换成sun-web.xml，修改的细节见后面章节
- 修改simpleTag.jsp，使用新的datasource名字
- 修改ExecuteSql.java，使用DataSource的资源注入
- 修改ExamplesHeader.jsp，使用glassfish的相对目录表示方式

修改之后的目录结构看起来像下面的样子：

```
GlassFish:  Exploded  EAR
jspSimpleTagEar
      META-INF
      ..  application.xml
      jspSimpleTagWar
      ..  ExamplesFooter.jsp
      ..  ExamplesHeader.jsp
      ..  SimpleTag.jsp
      ..  wls_examples.css
      ++  WEB-INF
            ..  web.xml
            ..  sun-web.xml
            ++  classes/examples/webapp/jsp/tags/simple
                  ..  ExecuteSql.class
                  ..  IteratorTag.class
            ++  tlds
                  ..  simpleTag.tld
```

所有的源代码修改完之后，我们可以用目录部署的方式来将修改后的应用部署到Glassfish

上

- asadmin deploydir gf_jspSimpleTag

Step 2: (Solaris) PointBase -> MySql

Weblogic 的例子使用自带的 PointBase 数据库，但是 Glassfish 目前还不支持 PointBase，所以我们使用 MySQL 来代替它。从这个步骤开始，我们所有的操作在 OpenSolaris 上进行。

Database name change

Weblogic 的例子所使用的数据库名字是 examples-dataSource-demoXAPool，我们这里把它改为 wls92examples。

Create and Populate wls92examples database

为了创建这个数据库，我们可以使用随 weblogic 一起发布的 mysql 脚本 demo_mysql.ddl。该脚本被打包在 server_pictures.zip 里面，是 weblogic windows 安装的一个子目录。

```
$ mysql -u root
mysql> create database wls92examples;
mysql> grant all on wls92examples.* to 'gengyong'@'localhost' identified by '12345678';
mysql> quit

$mysql -u gengyong -p
Enter password: 12345678
mysql> use wls92examples;
mysql> source
$BEA_INSTALL\weblogic92\samples\server\examples\src\examples\common\ddl\demo_mysql.ddl;
mysql> quit
```

Create a MySqlPool Connection Pool

现在我们回到 Glassfish Admin Console 来配置 MySQL 数据库连接池。需要创建的 JDBC 连接池是：

- 名称：MySqlPool
- 资源类型: javax.sql.DataSource
- 数据库厂商：MySQL

你可以在创建连接池的时候设置，也可以在创建以后通过以下路径来设置：

1. Resources -> Connection Pools -> MySqlPool
2. 选 AdditionalProperties 页

需要设置的属性有以下三个：

- RelaxAutoCommit = true 这是因为 ExecuteSql 类从 javax.servlet.jsp.tagext.TryCatchFinally 接口中实现了下面的一个方法

```
public void doCatch(Throwable t) throws Throwable {
        if (conn != null) conn.rollback();
        throw t;
}
```

这个方法假定 autocommit 是 false，否则上述方法就会产生一个 exception。

com.mysql.jdbc.exceptions.MySQLNonTransientConnectionException: Can't call rollback when autocommit=true

- user, password 属性。输入访问 wls92examples 数据库的用户名和密码：'gengyong'/'12345678'
- URL 属性。 jdbc:mysql://:3306/wls92examples 这个 URL 用来访问'wls92examples'数据库

Create a JDBC Resource jdbc/wls92examples

创建完了数据库连接池，我们可以来创建 JDBC 资源：

- JDBC name: jdbc/wls92examples
- Pool name: MySqlPool (上一步所创建的)
- Description:

Step 3: (Windows) Source Code Changes

由于数据库名字和访问方式的改变，我们的源代码需要作如下的修改：

SimpleTag.jsp: Data source name change

数据源名称的修改：

from

```
//  WebLogic
String  dataSourceName  =  "examples-dataSource-demoXAPool";
```

to

```
//  GlassFish
String  dataSourceName  =  "wls92examples;
```

ExecuteSql.java: Resource injection of DataSource

Weblogic 原来的例子在连接数据库的时候使用了不兼容的代码：

- 使用了 weblogic.jndi.WLInitialContextFactory
- JNDI 查找 t3://localhost:7001

为了使代码更加通用，我们可以使用 Java EE 5 的资源注入：

```
//File  name:  ExecuteSql.java
import  javax.sql.DataSource;
import  javax.annotation.Resource;

public  class  ExecuteSql  extends  BodyTagSupport  implements  TryCatchFinally  {
        ....
        @Resource(name="jdbc/wls92examples")
        DataSource  wlsds;
        ...  omitted  other  modifcations  to  code  ...
}
```

我们可以这么做，因为 ExecuteSql 是一个 managed 类。

为了重新编译这个 class，我们可以在 Netbeans 里面新建一个 Web 项目，以 Glassfish 作为应用服务器。新建这个类 ExecuteSql，注意包结构：examples.webapp.jsp.tags.simple。

Step 4: weblogic.xml -> sun-web.xml Mapping

我们用 sun-web.xml 来代替 weblogic.xml，它们是所谓的 application server specific 部署描述(deployment decriptor)文件。

```
<!-- weblogic.xml -->

  <?xml version="1.0" encoding="ISO-8859-1"?>
  <weblogic-web-app xmlns="http://www.bea.com/ns/weblogic/90">
  <jsp-descriptor>
    <page-check-seconds>1</page-check-seconds>
    <verbose>true</verbose>
  </jsp-descriptor>
    <!--
```

```
  Use the virtual-directory-mapping element to specify document roots other
  than the default document root of the Web application for certain kinds of
  requests, such as image requests. All images for a set of Web applications
  can be stored in a single location, and need not be copied to the document
  root of each Web application that uses them. For an incoming request, if a
  virtual directory has been specified servlet container will search for the
  requested resource first in the virtual directory and then in the
  Web application's original document root.
  -->
  <virtual-directory-mapping>
    <local-path>C:/bea/weblogic92/samples/server/</local-path>
    <url-pattern>/examples/*</url-pattern>
  </virtual-directory-mapping>
  <virtual-directory-mapping>
    <local-path>C:/bea/weblogic92/samples/server/examples/build</local-path>
    <url-pattern>images/*</url-pattern>
  </virtual-directory-mapping>


  <!-- sun-web.xml -->
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Application
Server 9.0 Servlet 2.5//EN" "http://www.sun.com/software/appserver/dtds/sun-
web-app_2_5-0.dtd">
  <sun-web-app>
    <jsp-config>
      <property name="modificationTestInterval" value="1"/>
      <property name="verbose" value="true"/>
    </jsp-config>

    <property name="alternatedocroot_1" value="from=/examples/*
dir=C:/bea/weblogic92/samples/server/"/>
    <property name="alternatedocroot_2" value="from=/images/*
dir=C:/bea/weblogic92/samples/server/examples/build"/>
  </sun-web-app>
```

(注：请把 windows 目录名 C:/bea/weblogic92/samples/server 修改成 server_pictures.zip 解压到 OpenSolaris 之后的目录名)

Virtual Directory Mapping

Weblogic 例子使用 virtual directory mapping 来寻找放在另外一个公共目录的 BEA logo 图片，Glassfish 相对应的功能是 alternatedocroot。

URL Pattern

另外，在 alternatedocroot_2 属性中，我们必须在 images/*前面加一个/，否则的话 server.log

里面会产生 Warning:

```
 [#|2008-05-15T17:57:49.500-0400|WARNING|sun-appserver9.1|
javax.enterprise.system.container.web|
_ThreadID=24;_ThreadName=httpWorkerThread-4848-1;images/*;_RequestID=1b29dda8-
9133-4863-b0ef-9b7ec44c8576;|WEB0504: URL pattern images/* for alternate
docbase is invalid|#]
```

Relative Path in URL

另外一个问题是关于 ExampleHeader.jsp 里面图片的相对路径：

```
<!-- weblogic : ExamplesHeader.jsp fragment -->
...

<!-- TITLE -->
<table border=0 cellspacing="18" cellpadding="0">
  <tr>
    <td valign="top">
      <a HREF="http://www.bea.com"><IMG
SRC="../../../../images/logo_tm_onwt.jpg" alt="BEA Logo" width="161"
height="96" border="0"></a>
      <h3><%=request.getParameter("title")%></h3>
    </td>
  </tr>
</table>
...
```

在 glassfish 里面，必须改成对 images/*的引用：

```
<!-- GlassFish : ExamplesHeader.jsp fragment -->
...

<table border=0 cellspacing="18" cellpadding="0">
  <tr>
    <td valign="top">
      <a HREF="http://www.bea.com"><IMG SRC="images/logo_tm_onwt.jpg" alt="BEA
Logo" width="161" height="96" border="0"></a>
      <h3><%=request.getParameter("title")%></h3>
    </td>
  </tr>
</table>
```

...

Verifier & Migrate2glassfish Tools

如果需要移植的应用规模比较大的话，我们也可以借助一些工具来完成这些 xml，jsp 文件的转换。

- **Verifier** – glassfish 自带的验证工具，位于$GLASSFISH_INSTALL/bin 目录下。每次部署一个应用到 glassfish，verifier 都会被用来检查所有部署描述符(deploy descriptor)的格式以及 JAVA EE API 的正确使用。
- **Migrate2glassfish** – 由 migrate2glassfish.dev.java.net 社区开发和维护一个 glassfish 的移植工具，帮助把 java,jsp,xml 等文件从其他应用服务器(JBoss, Weblogic, Websphere, TomCat 等)的格式转换成 glassfish 的格式。

Step 5: Deployment & Test

我们通过 asadmin 命令行工具来部署修改后的应用：

- asadmin deploydir gf_jspSimpleTag

然后通过浏览器访问： http://localhost:8080/jspsimpleTag/SimpleTag.jsp