在 Weblogic 下实现自动部署

周红

2004-8-13

目录

我们知道 Jbuilder 提供了对 J2EE 应用程序打包成.ear、.jar、.war 文件并可将其部署到应用服务器上的功能；使用 weblogic server admin console 也可以实现对这些文件的部署。不过当有很多的后台程序时一个一个部署就变得很麻烦，尤其是测试人员部署要测试的程序时。于是实现自动部署就显得格外重要。这几天我对 weblogic 的部署功能研究了一下，整理出来给大家共享。

一、Auto-Deployment

其实 weblogic 本身就有自动部署的功能。编辑 startWeblogic.cmd，设置 PRODUCTION_MODE=false，即为开发模式，然后我们直接把.ear、.jar、.war 文件放置在我们所建的 server 的 applications 目录下，Weblogic Server 系统就会自动部署这些应用；如果删除这些包，Weblogic Server 系统也会自动卸载，非常方便。不过本人试着更新一个已存在的包，Weblogic 好象不会自动加载它，除非重新启动 Weblogic，看来只好先删掉旧的再换上新的啦。大家不防也帮忙测试一下 weblogic 到底有没有这种 auto update 功能。

二、使用 weblogic.Deployer 命令行

% java weblogic.Deployer [options] [action] [files]
执行前要设置一下环境变量，运行 setEnv.cmd

介绍一下常用的[option]：

-adminurl <<protocol>://<server>:<port>> Administration server URL:
default iiop://localhost:7001
-username <username> user name
-password <password> password for the user
-examples Displays example usage of this tool.
-name <application name> Defaults to the basename of the deployment
file or directory.
-targets <<target(s)>> A comma separated list of targets for the
current operation. If not specified, all
configured targets are used. For a new
application, the default target is the
administration server.

常用的[action]：

-deploy Make an application available for service.
-redeploy Replace a running application partially or
entirely.
-undeploy Take an application out of service.

例如：

>java weblogic.Deployer -user weblogic -adminurl http://zhouh:7001 -password weblogic -name ejb11110 -deploy D:/tax/webback/tr11/tr11110/ejb11110.jar -targets myserver

使用 weblogic.Deployer utility 带来的好处是：

-通过选项可以更加灵活的部署和卸载应用程序

-实现远程部署

-可以同时对多个 weblogic server 部署

-写成批处理方便执行

例如：

```
set WL_HOME=C:\bea\weblogic81
set JAVA_HOME=C:\bea\jdk141_03
set CLASSPATH=%WL_HOME%\server\lib\weblogic.jar;
set STAGING= D:\tax\webback
%JAVA_HOME%\bin\java -classpath %CLASSPATH% weblogic.Deployer -user weblogic -adminurl http://appsvr:8001 -password weblogic -activate -name ejb11110 -upload -source %STAGING%\tr11\tr11110\ejb11110.jar -targets new_cluster_1
%JAVA_HOME%\bin\java -classpath %CLASSPATH% weblogic.Deployer -user weblogic -adminurl http://appsvr:8001 -password weblogic -activate -name tr11110 -upload -source %STAGING%\tr11\tr11110\tr11110.war -targets new_cluster_1
```

三、使用 Ant 构建和部署程序

1. 为什么使用 Ant？

每个被部署的单元，不管是 EJB JAR 文件、WAR 文件或 EAR 文件，都需要在特定的目录下放置部署描述符。正确的类必须被包含进来，并且在正确的目录中。对一个公用类的改变可能需要对整个应用重新部署。使用专有的 IDE，对于单个开发人员来说，能够使构建过程自动化，但当系统越来越复杂和庞大的时候，试图部署其他人的工作成果时，可能会是一场噩梦。

Ant 是一种专为 Java 量身订做的构建工具。Build 和 make 工具已经有了多年历史，他们通常用于应用的自动化构建过程。它们通常会运行编译器、访问文件系统以及处理与构建应用相关联的各种其他任务。这些工具一般和操作系统结合紧密。它们使用专有的文件格式，很难编辑、容易出错。虽然很多 make/build 工具可以用来构建 Java 应用，它们却不能提供操作系统的无关性，并且和 Java 的结合并不紧密。

Ant 是 Apache 的 Jakarta 项目的一部分，它集成了 java、javac、JAR、WAR 和其他一些 Java 任务。它也支持多种文件系统任务，获得了操作系统无关性。在 Windows 下编写的脚本也可以在 Solaris 下工作得很好。如果 Ant 不能满足您的要求，您可以使用 Java 来扩展它。因为 Ant 是可扩展的，它几乎可以和很多东西集成，包括 WLS。

Ant 没有替换 Java IDE，开发人员仍然使用他们熟悉的工具去开发代码和编写部署描述符，但当组件集成到要构建的项目中去的时候，应该包含一个 Ant build.xml 脚本。Build.xml 是描述应用或组件构建和部署过程的 xml 文件。

使用 Ant，您仍然需要做所有的配置工作，但是 Ant 将这个过程自动化了。Ant 允许您使用 XML 定义自己的打包、构建、部署过程。XML 文件成为一个活动的文档，开发组中的每个人都可以阅读和修改它，更重要的是每个人都可以执行它。这个过程可以通过更新 XML 构建脚本来不断求精。Ant 使得快速执行复杂的令人厌倦的操作而不出错误成为可能。

2.如何开始使用 Ant

对于 WLS6.1，Ant 已经包含在 weblogic.jar 包中。WLS 现在包含 Ant 的构件脚本和一些例子。BEA 在 WLS 的 bin 目录下有一个 Ant 的批处理和 Shell 脚本。换句话说，只要安装了 WLS6.1，Ant 自然也就安装了。假如您安装的是 6.1 以前版本的 WebLogic 服务器，只需要从以下地址下载最新版本的 Ant 并且解压：http://jakarta.apache.org/ant/manual/install.htm。解压后在 bin 目录下有 Ant 的批处理程序和脚本文件。

3. 运行 Ant

运行 Ant 非常简单，设置一下环境变量，path 填加 C:\bea\weblogic81\server\bin，然后运行 cmd 输入 ant 就可以了。

没有指定任何参数时，Ant 会在当前目录下查询 build.xml 文件。如果找到了就用该文件作为 buildfile。如果你用 -find 选项。Ant 就会在上级目录中寻找 buildfile，直至到达文件系统的根。要想让 Ant 使用其他的 buildfile，可以用参数 -buildfile file，这里 file 指定了你想使用的 buildfile。

可以指定执行一个或多个 target。当省略 target 时，Ant 使用标签<project>的 default 属性所指定的 target。

命令行选项总结：
ant [options] [target [target2 [target3] ...]]
Options:
-help print this message
-projecthelp print project help information
-version print the version information and exit
-quiet be extra quiet
-verbose be extra verbose
-debug print debugging information
-emacs produce logging information without adornments
-logfile file use given file for log output
-logger classname the class that is to perform logging
-listener classname add an instance of class as a project listener
-buildfile file use specified buildfile
-find file search for buildfile towards the root of the filesystem and use the first one found
-Dproperty=value set property to value

例子
ant
使用当前目录下的 build.xml 运行 Ant，执行缺省的 target。
ant -buildfile test.xml
使用当前目录下的 test.xml 运行 Ant，执行缺省的 target。
ant -buildfile test.xml dist
使用当前目录下的 test.xml 运行 Ant，执行一个叫做 dist 的 target。
ant -buildfile test.xml -Dbuild=build/classes dist
使用当前目录下的 test.xml 运行 Ant，执行一个叫做 dist 的 target，并设定 build 属性的值为 build/classes。

4 编写 build.xml

Ant 的 buildfile 是用 XML 写的。每个 buildfile 含有一个 project。

buildfile 中每个 task 元素可以有一个 id 属性，可以用这个 id 值引用指定的任务。这个值必须是唯一的。（详情请参考下面的 Task 小节）

4.1 Projects

project 有下面的属性：
Attribute Description Required
name 项目名称. No
default 当没有指定 target 时使用的缺省 target Yes
basedir 用于计算所有其他路径的基路径。该属性可以被 basedir property 覆盖。当覆盖时，该属性被忽略。如果属性和 basedir property 都没有设定，就使用 buildfile 文件的父目录。 No
项目的描述以一个顶级的<description>元素的形式出现（参看 description 小节）。

一个项目可以定义一个或多个 target。一个 target 是一系列你想要执行的。执行 Ant 时，你可以选择执行那个 target。当没有给定 target 时，使用 project 的 default 属性所确定的 target。

4.2 Targets

一个 target 可以依赖于其他的 target。例如，你可能会有一个 target 用于编译程序，一个 target 用于生成可执行文件。你在生成可执行文件之前必须先编译通过，所以生成可执行文件的 target 依赖于编译 target。Ant 会处理这种依赖关系。

然而，应当注意到，Ant 的 depends 属性只指定了 target 应该被执行的顺序－如果被依赖的 target 无法运行，这种 depends 对于指定了依赖关系的 target 就没有影响。

Ant 会依照 depends 属性中 target 出现的顺序（从左到右）依次执行每个 target。然而，要记住的是只要某个 target 依赖于一个 target，后者就会被先执行。
<target name="A"/>
<target name="B" depends="A"/>

<target name="C" depends="B"/>
<target name="D" depends="C,B,A"/>
假定我们要执行 target D。从它的依赖属性来看，你可能认为先执行 C，然后 B，最后 A 被执行。错了，C 依赖于 B，B 依赖于 A，所以先执行 A，然后 B，然后 C，最后 D 被执行。

一个 target 只能被执行一次，即时有多个 target 依赖于它（看上面的例子）。

如果(或如果不)某些属性被设定,才执行某个 target。这样,允许根据系统的状态(java version, OS, 命令行属性定义等等）来更好地控制 build 的过程。要想让一个 target 这样做，你就应该在 target 元素中，加入 if（或 unless）属性，带上 target 因该有所判断的属性。例如：
<target name="build-module-A" if="module-A-present"/>
<target name="build-own-fake-module-A" unless="module-A-present"/>
如果没有 if 或 unless 属性，target 总会被执行。

可选的 description 属性可用来提供关于 target 的一行描述，这些描述可由-projecthelp 命令行选项输出。

将你的 tstamp task 在一个所谓的初始化 target 是很好的做法，其他的 target 依赖这个初始化 target。要确保初始化 target 是出现在其他 target 依赖表中的第一个 target。在本手册中大多数的初始化 target 的名字是"init"。

target 有下面的属性：
Attribute Description Required
name target 的名字 Yes
depends 用逗号分隔的 target 的名字列表，也就是依赖表。 No
if 执行 target 所需要设定的属性名。 No
unless 执行 target 需要清除设定的属性名。 No
description 关于 target 功能的简短描述。 No

4.3 Tasks

一个 task 是一段可执行的代码。

一个 task 可以有多个属性（如果你愿意的话，可以将其称之为变量)。属性只可能包含对 property 的引用。这些引用会在 task 执行前被解析。

下面是 Task 的一般构造形式：
<name attribute1="value1" attribute2="value2" ... />
这里 name 是 task 的名字，attributeN 是属性名，valueN 是属性值。

有一套内置的（built-in）task，以及一些可选 task，但你也可以编写自己的 task。

所有的 task 都有一个 task 名字属性。Ant 用属性值来产生日志信息。

可以给 task 赋一个 id 属性：
<taskname id="taskID" ... />
这里 taskname 是 task 的名字，而 taskID 是这个 task 的唯一标识符。通过这个标识符，你可以在脚本中引用相应的 task。例如，在脚本中你可以这样：
<script ... >
task1.setFoo("bar");
</script>
设定某个 task 实例的 foo 属性。在另一个 task 中（用 java 编写），你可以利用下面的语句存取相应的实例。
project.getReference("task1").
注意 1：如果 task1 还没有运行，就不会被生效（例如：不设定属性），如果你在随后配置它，你所作的一切都会被覆盖。

注意 2：未来的 Ant 版本可能不会兼容这里所提的属性，因为很有可能根本没有 task 实例，只有 proxies。

4.4 Properties

一个 project 可以有很多的 properties。可以在 buildfile 中用 property task 来设定，或在 Ant 之外设定。一个 property 有一个名字和一个值。property 可用于 task 的属性值。这是通过将属性名放在"${"和"}"之间并放在属性值的位置来实现的。例如如果有一个 property builddir 的值是"build"，这个 property 就可用于属性值：${builddir}/classes。这个值就可被解析为 build/classes。

内置属性

如果你使用了<property> task 定义了所有的系统属性，Ant 允许你使用这些属性。例如，${os.name}对应操作系统的名字。

要想得到系统属性的列表可参考 the Javadoc of System.getProperties。

除了 Java 的系统属性，Ant 还定义了一些自己的内置属性：
basedir project 基目录的绝对路径 (与<project>的 basedir 属性一样)。
ant.file buildfile 的绝对路径。
ant.version Ant 的版本。
ant.project.name 当前执行的 project 的名字；由<project>的 name 属性设定.
ant.java.version Ant 检测到的 JVM 的版本； 目前的值有"1.1", "1.2", "1.3" and "1.4".

例子
<project name="MyProject" default="dist" basedir=".">

<!-- set global properties for this build -->
<property name="src" value="."/>
<property name="build" value="build"/>

```
<property name="dist" value="dist"/>

<target name="init">
<!-- Create the time stamp -->
<tstamp/>
<!-- Create the build directory structure used by compile -->
<mkdir dir="${build}"/>
</target>

<target name="compile" depends="init">
<!-- Compile the java code from ${src} into ${build} -->
<javac srcdir="${src}" destdir="${build}"/>
</target>

<target name="dist" depends="compile">
<!-- Create the distribution directory -->
<mkdir dir="${dist}/lib"/>
<!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
<jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
</target>

<target name="clean">
<!-- Delete the ${build} and ${dist} directory trees -->
<delete dir="${build}"/>
<delete dir="${dist}"/>
</target>

</project>
```

4.5 Path-like Structures

你可以用":"和";"作为分隔符，指定类似 PATH 和 CLASSPATH 的引用。Ant 会把分隔符转换为当前系统所用的分隔符。

当需要指定类似路径的值时，可以使用嵌套元素。一般的形式是

```
<classpath>
<pathelement path="${classpath}"/>
<pathelement location="lib/helper.jar"/>
</classpath>
```

location 属性指定了相对于 project 基目录的一个文件和目录，而 path 属性接受逗号或分号分隔的一个位置列表。path 属性一般用作预定义的路径－－其他情况下，应该用多个 location 属性。

关于 ant 的详细使用大家可到 http://ant.apache.org/manual/index.html 查阅。

5. 应用到我们系统

5.1 部署一个应用

下面就我们的系统，写一个 build.xml 例子，这个 build.xml 实现了对 43811 的编译和部署，放置在 D:\tax\webback\tr43\tr43811 目录下。

```
<?xml version="1.0"?>
<project name="tr43811" default="43811" basedir=".">
    <property name="src" value="src/tr43811"/>
    <property name="build" value="classes"/>
    <property name="lib" value="lib"/>
    <property name="jarname" value="ejb43811.jar"/>
    <property name="warname" value="tr43811.war"/>
    <property name="earname" value="ear43811.ear"/>
    <property name="appname" value="ear43811"/>
    <property name="web_inf" value="./tr43811/WEB-INF"/>
    <property name="meta_inf" value="./META-INF"/>
    <property name="WL_HOME" value="C:\bea\weblogic81"/>
    <property name="adminurl" value="t3://localhost:7001"/>
    <property name="server" value="myserver"/>
    <property name="username" value="weblogic"/>
    <property name="password" value="weblogic"/>

    <target name="init" depends="clean">
        <mkdir dir="${build}"/>
        <mkdir dir="${lib}"/>
    </target>

    <target name="clean">
        <delete dir="${build}"/>
        <delete dir="${lib}"/>
        <delete file="${jarname}"/>
        <delete file="${warname}"/>
        <delete file="${earname}"/>
    </target>

    <target name="compile" depends="init">
        <javac srcdir="${src}" destdir="${build}">
```

```
                <classpath>
                    <pathelement path=""/>
                    <pathelement location="D:/tax/webback/wwcom/wwcom.jar"/>
                    <pathelement location="D:\tax\webback\lib\jdom.jar"/>
                    <pathelement location="C:/oracle/ora90/jdbc/lib/classes12.jar"/>
                </classpath>
            </javac>
            <mkdir dir="${build}/META-INF"/>
            <copy todir="${build}/META-INF">
        <fileset dir="${meta_inf}"/>
        </copy>
        </target>

        <target name="jar" depends="compile">
            <jar jarfile="${jarname}" >
                <fileset dir="${build}"/>
            </jar>
        </target>

        <target name="war" depends="compile">
            <war destfile="${warname}" webxml="${web_inf}/web.xml">
                <fileset dir="${src}"/>
            </war>
        </target>
        <target name="ear" depends="jar,war">
            <ear destfile="${earname}" appxml="application.xml">
                <fileset dir="." includes="*.jar,*.war"/>
            </ear>
        </target>

        <target name="deploy" depends="ear">
            <taskdef                                                    name="wldeploy"
classname="weblogic.ant.taskdefs.management.WLDeploy"/>
            <wldeploy action="deploy"
              source="${earname}"
              name="${appname}"
              verbose="true"
              adminurl="${adminurl}"
              user="${username}"
              password="${password}"
              targets="${server}"
              debug="true"/>

        </target>
```

```
    <target name="43811" depends="deploy"/>
</project>
```

这个 build.xml 完成了这样一些目标：
首先定义了一些属性，因为在每个人不同开发环境可能会有所变化。
然后做一些初始准备：删除过时的目录，建立新的目录。
接下来对 src 目录下的所有 java 类进行编译，并将部署描述符拷贝到 build 目录。
然后创建 Jar 文件。
然后创建 war 文件。
最后发布到服务器上。

运行 ant 命令：D:\tax\webback\tr43\tr43811>ant 43811
tr43811 就被编译并部署到服务器上。

使用 ant 也有缺点，我们就要手工编写 ejb-jar.xml、weblogic-ejb-jar.xml，application.xml，而不象 jbuild 等 IDE 工具可以自动生成部署描述符。当然这不是问题，找个现成的改改就好了。其实 Jbuild 生成的.eargrp，.ejbgrpx 文件就是部署描述符，可拿来修改一下。

5.2 让大型应用的部署变得容易

分开运行每个 build.xml 脚本文件很麻烦，我们最终目的是要一次性部署所有组件。我们可以为应用创建一个运行所有 Ant 构建脚本的 Ant 构建脚本。这个脚本将应用中所有的脚本组合在一起。我们定义一个 all 目标，他本身没用具体功能，只是调用所有的依赖。

```
<?xml version="1.0"?>
<project name="TAMIS" default="all" basedir=".">
  <target name="all">
    <ant antfile="build.xml" dir="D:\tax\webback\tr11\tr11110\"/>
    <ant antfile="build.xml" dir="D:\tax\webback\tr11\tr11211\"/>
    <ant antfile="build.xml" dir="D:\tax\webback\tr11\tr11251\"/>
    ……
  </target>
 </project>
```

每个开发人员将每个模块的源代码和它的 build.xml 文件一并提交，放到版本控制系统中。如果有一个公共的组件被修改，我们只需执行总 Ant 构建脚本即可轻松完成后台所有组件的编译和部署。