# 打包开箱即用（OOBE）版BPMX

BPMX标准的JavaWeb应用程序，安装布署时，需要准备好JDK、支持的关系数据库和Servlet2.4规范的WEB服务器。

## 解决方案

打包OOBE BPMX，需要将jdk、数据库、web服务器与bpmx应用一起打包。最终的发布包的可用性与大小与选择的数据库、web服务器相关。嵌入式数据库和嵌入式tomcat组合可以满足要求。

通过一个建立打包应用程序，由打包应用程序完成相应的工作。

### 数据库

选择的数据库为H2。

### web服务器

选择的Web服务器为嵌入式Tomcat。

Tomcat

嵌入式Tomcat的Maven依赖为：

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-core</artifactId>

<version>${tomcat.embed.version}</version>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<version>${tomcat.embed.version}</version>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-logging-juli</artifactId>

<version>${tomcat.embed.version}</version>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-logging-log4j</artifactId>

<version>${tomcat.embed.version}</version>

</dependency>

使用嵌入式Tomcat布署Web应用程序的流程与独立使用的Tomcat类似。但区别是使用嵌入式Tomcat时，所有的步骤与参数设置都是通过Java代码进行控制。

在使用独立版本Tomcat时，我们知道，Tomcat被模块化为一系统有着各自目的的组件，这些组件的配置通过server.xml文件进行配置和组装。当使用embed tomcat时，我们无法使用server.xml文件，所以我们必须以编程的方式对tomcat进行参数设置的组装。embed tomcat提供了一系列的core interface，允许我们组织这些组件，装配一个可用的tomcat实例。Tomcat主要的类与结构组件的关系如表-1所示

|  |  |  |
| --- | --- | --- |
| 接口/类 | 组件 | 描述 |
| Interface org.apache.catalina.Server | Server | 整个Tomcat实例，包含一个或多个Services |
| Interface org.apache.catalina.Service | Service | Engine的容器 ，通过Connector进行访问。 |
| Class org.apache.catalina.Connector | Connector | 定义到Tomcat实例的连接 |
| Interface org.apache.catalina.Engine | Engine | Servlet 引擎 |
| Interface org.apache.catalina.Host | Host | 虚拟主机 |
| Interface org.apache.catalina.Context | Context | 表示包含在Host中的独立的一个web应用 |

表-1

表-1中所有的Tomcat定义的接口，Tomcat都提供了一个或多个实现，我们可直接用于构建tomcat实例。

embed tomcat提供了一个方便构建tomcat实例的类org.apache.catalina.startup.Tomcat。

使用类org.apache.catalina.startup.Tomcat，通过下面简单的代码即可构建一个embed tomcat实例：

**package** com.hotent;

**import** org.apache.catalina.startup.Tomcat;

**import** org.apache.catalina.core.StandardServer;

**import** org.apache.catalina.core.AprLifecycleListener;

**import** org.apache.catalina.LifecycleException;

**import** javax.servlet.ServletException;

**public** **class** EmbeddedTomcat {

**private** Tomcat tomcat;

**private** **void** startTomcat() **throws** ServletException, LifecycleException {

**this**.tomcat = **new** Tomcat();

//设置tomcat监听端口为8080

**this**.tomcat.setPort(8080);

//设置基目录

**this**.tomcat.setBaseDir(".");

StandardServer server = (StandardServer) **this**.tomcat.getServer();

//添加生命周期监听事件

AprLifecycleListener listener = **new** AprLifecycleListener();

server.addLifecycleListener(listener);

//应用程序的上下文路径和应用所在的系统路径

String contextPath = "/myapp";

String appBase = "/opt/tomcat7/webapps/examples";

**this**.tomcat.addWebapp(contextPath, appBase);

**this**.tomcat.start();

}

**private** **void** stopTomcat() **throws** LifecycleException {

**this**.tomcat.stop();

}

**public** **static** **void** main(String args[]) {

**try** {

EmbeddedTomcat tomcat = **new** EmbeddedTomcat();

//启动tomcat实例

tomcat.startTomcat();

Thread.*sleep*(100000);

//停止tomcat实例

tomcat.stopTomcat();

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

## 实现

### 打包器-程序结构图



【启动器】的工作的主要工作是创建和初始化H2数据库、创建和初始化Tomcat对象。

【编译打包BPMX应用程序】的工作是使用ANT对BPMX应用程序的源码进行编译，生成可部署的WEB应用。由Maven的maven-antrun-plugin插件实现。

<plugin>

<artifactId>maven-antrun-plugin</artifactId>

<version>1.7</version>

<executions>

<execution>

<phase>package</phase>

<configuration>

<target>

<ant antfile=*"ant-maven-build.xml"*/>

</target>

</configuration>

<goals>

<goal>run</goal>

</goals>

</execution>

</executions>

</plugin>

【构建可执行程序】的工作是生成可执行的应用程序目录结构和BASH SHELL 和DOS脚本、配置文件等，但生成的程序还不能独立运行，因为一些资源还没有完成准备好，还需要进一步的工作。由Maven的appassembler-maven-plugin插件实现。

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>appassembler-maven-plugin</artifactId>

<version>1.3</version>

<configuration>

<!-- Set the target configuration directory to be used in the bin scripts -->

<configurationDirectory>conf</configurationDirectory>

<!-- Copy the contents from "/src/main/config" to the target configuration directory in the assembled application -->

<copyConfigurationDirectory>true</copyConfigurationDirectory>

<!-- Include the target configuration directory in the beginning of the classpath declaration in the bin scripts -->

<includeConfigurationDirectoryInClasspath>true</includeConfigurationDirectoryInClasspath>

<!-- set alternative assemble directory -->

<assembleDirectory>${project.build.directory}/app</assembleDirectory>

<!-- Generate bin scripts for windows and unix pr default -->

<platforms>

<platform>windows</platform>

<platform>unix</platform>

</platforms>

<!-- Extra JVM arguments that will be included in the bin scripts -->

<extraJvmArguments>

-Xms128m -Xmx512m -XX:PermSize=128M -XX:MaxNewSize=128m -XX:MaxPermSize=512m

</extraJvmArguments>

<repositoryLayout>flat</repositoryLayout>

<includeConfigurationDirectoryInClasspath>true</includeConfigurationDirectoryInClasspath>

<windowsScriptTemplate>src/main/assembly/windowsBinTemplate\_packJdk</windowsScriptTemplate>

<unixScriptTemplate>src/main/assembly/unixBinTemplate\_packJdk</unixScriptTemplate>

<programs>

<program>

<mainClass>com.hotent.Main</mainClass>

<name>bpmx-starter</name>

</program>

</programs>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>assemble</goal>

</goals>

</execution>

</executions>

</plugin>

【Assemble打包程序】的工作是将前面的步骤生成的资源最终assemble为可使用的最终OOBE程序包。由Maven的maven-assembly-plugin插件实现。

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-assembly-plugin</artifactId>

<version>2.3</version>

<configuration>

<descriptor>src/main/assembly/bin.xml</descriptor>

<finalName>Starter</finalName>

<outputDirectory>assembler</outputDirectory>

<appendAssemblyId>false</appendAssemblyId>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>single</goal>

</goals>

</execution>

</executions>

</plugin>