



IBM WebSphere Liberty 安装配置

开发管理使用指南

国际商业机器（中国）有限公司广州分公司

俞黎敏

YuLimin@cn.ibm.com

2018年12月07日

目录

1. 总体介绍.....	5
1.1. 背景信息.....	5
1.2. 概述.....	5
2. 下载 Liberty.....	7
3. 极简安装启动部署一站式体验.....	8
3.1. 解压安装.....	8
3.1.1. 执行解压.....	8
3.1.2. 配置 Java 环境.....	9
3.2. 创建服务器并启停.....	10
3.3. 部署应用.....	11
4. 扩展功能.....	11
5. 开发调试环境搭建.....	12
5.1. 本地文件安装.....	13
5.2. 在线安装.....	14
6. Eclipse 中配置使用 Liberty 服务器.....	17
6.1. 配置 Liberty 服务器.....	17
6.2. 创建 Web 项目并调试.....	20
6.3. Maven.....	23
6.3.1. 插件仓库.....	23
6.3.2. 服务器的管理.....	24
6.3.3. 部署或删除应用程序.....	24
6.3.4. Maven 私服搭建.....	24
6.4. ANT.....	25
6.4.1. 服务器的管理.....	25
6.4.2. 部署或删除应用程序.....	25
7. Liberty 应用开发.....	26
7.1. 简单应用程序开发.....	26
7.1.1. JSP、Servlet 应用程序开发.....	26
7.1.2. JAX-RS RESTful Web Services 应用程序开发.....	26
7.1.3. CDI 应用程序开发.....	26
7.1.4. JAX-WS Web Services 应用程序开发.....	26
7.1.5. WebSocket 应用程序开发.....	26
7.2. OSGi 应用程序开发.....	27
7.3. 企业应用程序开发.....	28
7.3.1. 数据库访问应用程序开发.....	28
7.3.2. EJB 应用程序开发.....	28
7.3.3. JMS 应用程序开发.....	28
7.3.4. 异步、并发应用程序开发.....	28
7.3.5. JavaMail 邮件应用程序开发.....	28

8.	管理控制台安装使用.....	29
8.1.	安装 adminCenter 组件.....	29
8.2.	添加 adminCenter 配置.....	31
8.3.	启动 adminCenter 服务.....	33
8.4.	使用管理控制台.....	36
8.5.	操作应用程序.....	37
8.6.	操作服务器.....	38
8.7.	查看 JVM 信息.....	39
9.	生产环境配置.....	40
9.1.	JDK 配置.....	40
9.1.1.	系统 JAVA_HOME 配置 JDK.....	40
9.1.2.	通过 server.env 配置 JDK.....	40
9.1.3.	通过 jvm.options 配置 JVM 参数.....	40
9.2.	共享库配置.....	41
9.3.	安全配置.....	42
9.3.1.	生成证书.....	42
9.3.2.	配置 Liberty.....	49
9.3.3.	启动 Liberty 加载新配置.....	51
9.3.4.	通过 Eclipse 开发工具配置.....	52
9.4.	集群配置与管理.....	55
9.4.1.	创建集合 (Collective).....	57
9.4.1.1.	创建集合控制器服务器.....	57
9.4.1.2.	创建集合控制器配置信息.....	57
9.4.1.3.	启动控制器服务器.....	60
9.4.1.4.	创建成员并加入集合.....	63
9.4.2.	执行集合操作.....	66
9.5.	集群规划与创建.....	70
9.5.1.	创建集合成员.....	71
9.5.2.	将集合成员分配到集群.....	74
9.6.	构建高可用的集合控制器.....	77
9.7.	部署应用到集群.....	86
9.8.	生成插件.....	89
9.9.	分布式缓存.....	91
9.10.	生产环境调优.....	92
9.10.1.	禁用应用程序监控.....	92
9.10.2.	禁用配置文件监控.....	92
9.11.	生产环境最佳实践.....	93
9.12.	问题诊断.....	94
9.12.1.	日志和跟踪.....	94
9.12.2.	服务器内存 Dump.....	96
9.12.3.	MBeans 及 JConsole.....	98
9.12.4.	OSGi 调试控制台.....	101
9.12.4.1.	启用 OSGi 控制台.....	101
9.12.4.2.	配置 OSGi Bundle 仓库.....	102

9.12.5.	事件日志.....	103
9.12.6.	请求访问慢以及线程挂起监测.....	104
9.12.7.	时间操作.....	105
9.12.8.	如何定义整个服务器的 404、50X 之类的页面?	106
10.	Docker 支持.....	107
10.1.	Java EE7 & Docker 支持.....	107
10.2.	Docker 安装.....	107
10.2.1.	Windows 平台下 Docker 安装.....	107
10.2.2.	Linux 平台下 Docker 安装	107
10.2.3.	Mac 平台下 Docker 安装.....	107
10.3.	Docker 私服构建.....	107
10.4.	下载 Liberty 镜像.....	108
10.5.	执行并运行应用程序.....	108
10.6.	自定义镜像.....	108
10.7.	Docker 下 Liberty 集群构建	109
11.	运行环境配置迷你化.....	110
12.	启用调试并进行远程调试.....	111
13.	数据库连接池配置示例.....	112
13.1.	DB2 数据库	112
13.2.	Oracle 数据库.....	112
13.3.	MySQL 数据库.....	113
13.4.	Derby 数据库.....	113
14.	后记.....	115
15.	资料参考.....	116

1. 总体介绍

1.1. 背景信息

IBM WebSphere Application Server 向来以重量级而著称，而大量抛弃 EJB 这大巨头后，无论商用还是开源的应用服务器都走上轻量化的轨道。IBM 也推出了 IBM WebSphere Application Server Liberty Profile 来应对并争取更大的开发者市场。但 IBM WebSphere Application Server 其他版本仍旧是行走在重量级的大道上。

IBM WebSphere Liberty 轻量化企业级应用服务器也是应云时代而生的应用服务器。麻雀虽小，五脏俱全。虽然 IBM WebSphere Liberty 体积很小，但具备的内容却很齐全，完全遵循 Java EE 最新规范进行实现。

1.2. 概述

IBM WebSphere Liberty 安装文件很小，这了让用户有更多的选择以及体现其组件化积木式的功能特性，它提供了多种的版本供用户下载，按功能多少与体积大小来分：

实现 **Java EE 8** 规范相关的软件包：

- Liberty Kernel：这是 Liberty 最基本的运行时，但是没有包含功能特性，需要开始通过 bin/installUtility 工具来“积木式”地按需安装相应的功能。文件大小约 12M。
- Liberty with Java EE 8 Web Profile：这是通过 Java EE 8 Web Profile 认证的包，在日常开发与运行环境中基本上满足要求。文件大小约 80M。
- Liberty with Java EE 8 Full Platform：这是通过 Java EE 8 规范认证的包，当你在开始与运行环境中需要完整的 Java EE 8 规范实现时，则需要下载此包。文件大小约 80M。
- Liberty with Java EE 8 Web Profile and IBM Java SDK 8：这是通过 Java EE 8 Web Profile 认证的包，并且包含了 IBM Java SDK 8 的包，在日常开发与运行环境中完全满足要求，如果你的环境中没有包含 Java SDK，那么，这个下载包是最合适不过了。文件大小根据平台的不同而异，Windows 平台大概约在 260M，Linux X86 平台约在 220M，还提供有 Linux on Power PC (64 bit)、Linux on Power PC Little Endian、Linux on z Systems (64 bit)版本让用户根据自己的生产环境操作平台进行灵活地选择。
- Liberty with OSGi Applications：这是专门用来支持 OSGi 应用程序开发与运行的包。文件大小约为 69M。
- Liberty with MicroProfile for enterprise Java：这是专门用来支持企业 Java MicroProfile 应用程序开发与运行的包。文件大小约 46M。
- Liberty with Java EE 8 Application Client：这是包含了 Java EE 8 Client 客户端所需

要的程序包。文件大小约为 62M。

还有实现 **Java EE 7** 规范相关的软件包：

- Liberty with Java EE 7 Web Profile：这是通过 Java EE 7 Web Profile 认证的包，在日常开发与运行环境中基本上满足要求。文件大小约 80M。
- Liberty with Java EE 7 Full Platform：这是通过 Java EE 7 规范认证的包，当你在开始与运行环境中需要完整的 Java EE 7 规范实现时，则需要下载此包。文件大小约 80M。
- Liberty with Java EE 7 Web Profile and IBM Java SDK 8：这是通过 Java EE 7 Web Profile 认证的包，并且包含了 IBM Java SDK 8 的包，如果在日常开发与运行环境中仅需要 Java EE 7 规范相关的技术要求，则这个包可以完全满足你的要求，如果你的环境中没有包含 Java SDK，那么，这个下载包是最合适不过了。文件大小根据平台的不同而异，Windows 平台大概约在 260M，Linux X86 平台约在 220M，同样还提供有 Linux on Power PC (64 bit)、Linux on Power PC Little Endian、Linux on z Systems (64 bit)版本让用户根据自己的生产环境操作平台进行灵活地选择。
- Liberty with Java EE 7 Application Client：这是包含了 Java EE 7 Client 客户端所需要的程序包。文件大小约为 59M。
-

以及 Java EE 7 规范相关的软件包：

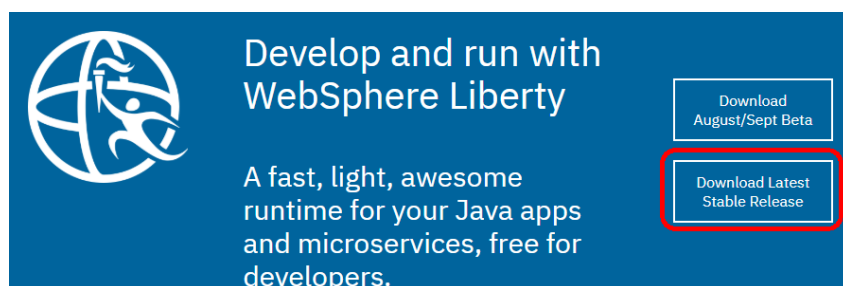
- Liberty Runtime：这是通过 Java EE 6 Web Profile 认证的包，在日常开发与运行环境中基本上满足要求。文件大小约 69M。

在这个绿色环保、苗条为上的大主题环境下，或许这个瘦身版本的安装文件尺寸就成为打动用户极为重要的一个敲门砖。

IBM WebSphere Application Server Liberty Profile 的出发点主要是为了快速开发 Web 应用、OSGi 应用、Web 2.0 以及 Mobile 应用，主要以轻量化为主；通过 OSGi 动态加载的方式实现不同功能模块的动态配置与加载，实现热插拔式的功能。

2. 下载 Liberty

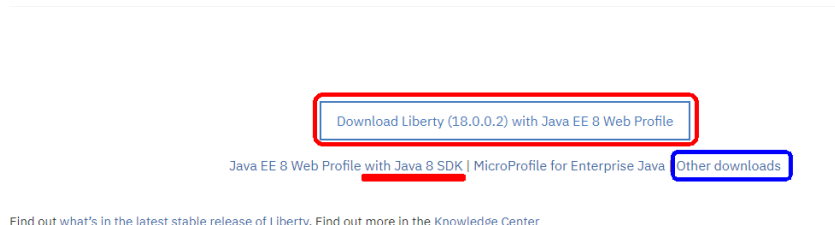
通过浏览器访问 <http://wasdev.net> 可以看到最新的测试版本与最新的正式发行版本下载链接



对于想体验最新 Java EE 新特性以及 Liberty 新功能特性的开发者则可以直接下载最新的 Beta 版本进行体验。为了开发与生产环境进行运行则下载稳定版本 Stable Release 即可。

同时，在 IBM 的官网上还提供了带有 Java SDK 的完整的下载包，供用户者选择，方便使用，如下图所示：

Download the latest stable WebSphere Liberty runtime



通过“Other downloads”链接可以根据需要找到更多可以下载的软件与工具等等。

我们下载 Liberty with Java EE 8 Web Profile and IBM Java SDK 8：这是通过 Java EE 8 Web Profile 认证的包，并且包含了 IBM Java SDK 8 的包。

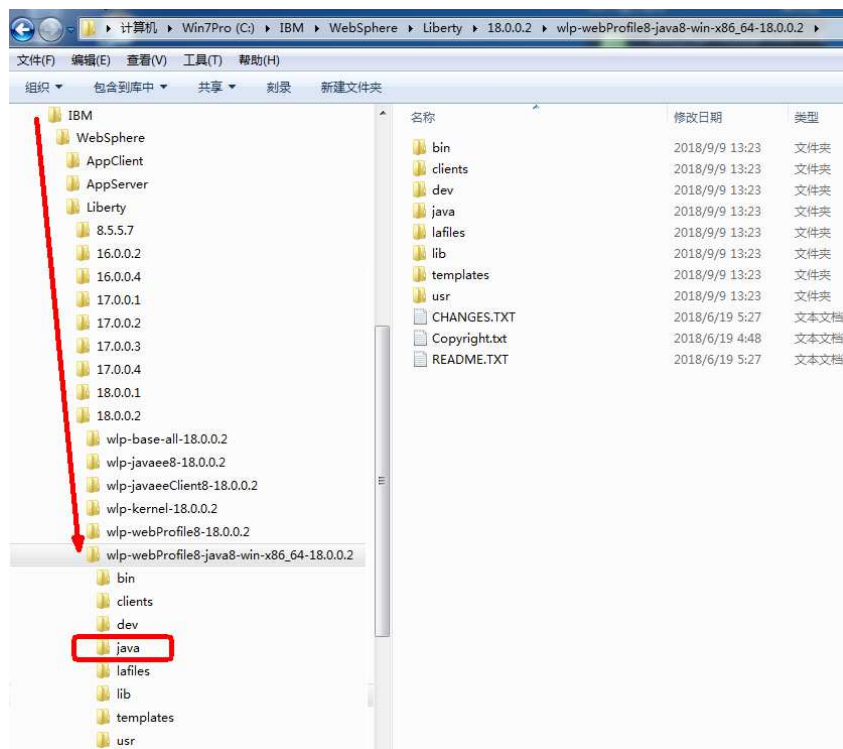
3. 极简安装启动部署一站式体验

3.1. 解压安装

3.1.1. 执行解压

将下载好 Liberty 包 wlp-webProfile8-java8-win-x86_64-18.0.0.2.zip 文件进行进行解压出来即可，比如解压到如下目录，目录大家可以自己决定：

C:\IBM\WebSphere\Liberty\18.0.0.2\wlp-webProfile8-java8-win-x86_64-18.0.0.2



Linux 等系统下，也同样进行解压，比如解压到如下目录：

/opt/IBM/WebSphere/Liberty/18.0.0.2/wlp-webProfile8-java8-win-x86_64-18.0.0.2

3.1.2. 配置 Java 环境

由于我们下载是自带有 Java SDK 的，所以可以直接用自带的 Java SDK，如果下载的是其他版本或者想用自己已有的 Java SDK，则同样进行配置系统环境变量 `JAVA_HOME` 与 `PATH` 即可。

Liberty 可以支持 IBM、SUN 等 JDK，在安装与运行之前先进行指定与配置 `JAVA_HOME` 和 `PATH`，比如：

Windows 命令行配置自带的 IBM JDK：

```
set WLP_HOME=C:\IBM\WebSphere\Liberty\18.0.0.2\wlp-webProfile8-java8-win-x86_64-18.0.0.2
set JAVA_HOME=%LIBERTY_HOME%\java
set PATH=%JAVA_HOME%\bin;%PATH%
```

Windows 命令行配置 SUN 的 JDK：

```
set JAVA_HOME=D:\JDK\SUN\8.0
set PATH=%JAVA_HOME%\bin;%PATH%
```

或者配置 IBM WAS 9.0 ND 带的 JDK

```
set JAVA_HOME=D:\IBM\WebSphere\AppServer\9.0\java
set PATH=%JAVA_HOME%\bin;%PATH%
```

Linux 下同样可以进行配置环境变量

Linux 命令行配置自带的 IBM JDK：

```
export WLP_HOME=/opt/IBM/WebSphere/Liberty/18.0.0.2/wlp-webProfile8-java8-win-x86_64-18.0.0.2
export JAVA_HOME=$LIBERTY_HOME/java
export PATH=$JAVA_HOME/bin:$PATH
```

3.2. 创建服务器并启停

切换到 Liberty 的主目录的可执行 bin 目录中去，为了方便今后的操作，可以设置环境变量 **WLP_HOME** 指向 Liberty 的主目录

```
set WLP_HOME=C:\IBM\WebSphere\Liberty\18.0.0.2\wlp-webProfile8-java8-win-x86_64-18.0.0.2
```

```
cd /d %WLP_HOME%\bin
```

然后，创建默认服务器并启动
直接通过运行

server start

即可以创建默认的 defaultServer 并启动了。

```
C:\IBM\WebSphere\Liberty\18.0.0.2\wlp-webProfile8-java8-win-x86_64-18.0.0.2\bin>server start
```

正在启动服务器 defaultServer。
服务器 defaultServer 已启动。

通过执行

server stop

即可以停止默认的 defaultServer 了

```
C:\IBM\WebSphere\Liberty\18.0.0.2\wlp-webProfile8-java8-win-x86_64-18.0.0.2\bin>server stop
```

正在停止服务器 defaultServer。
服务器 defaultServer 已停止。

查看服务器版本信息

server version

输出如下信息

```
IBM J9 VM Vpwa6480sr3-20160428_01 (SR3) (zh_CN) 上的 WebSphere Application Server
18.0.0.2 (1.0.21.cl180220180619-0403)
```

3.3. 部署应用

接下来，开始部署应用，简单，通过拖拽的方式就可以完成部署与启动，将 .war 包或者 .ear 包直接放进对应服务器的 dropins 目录中即可部署并启动，而且无需要重新启动服务器即可完成应用程序的部署，默认目录为如下目录：

%WLP_HOME%\usr\servers\defaultServer\dropins

4. 扩展功能

根据所解压的 Liberty 安装包决定了它所包含了那些具体的功能包，如果需要 Web Service、JMS 或 MongoDB 等各种新技术的直接支持，并且它们没有包含在所下载的解压包当中的，则需要通过 installUtility 工具来安装 Liberty 的功能包。

installUtility find <查找包名称>

比如，执行

installUtility find jsp

正在建立与已配置存储库的连接...
此过程可能要花几分钟完成。

已成功连接至所有已配置的存储库。

正在搜索资产。此过程可能要花几分钟完成。

feature : jsp-2.2 : JavaServer Pages 2.2
feature : jsp-2.3 : JavaServer Pages 2.3
sample : cdiSample : Contexts and Dependency Injection (CDI) Sample
sample : OnlinePollingSample : EJB in WAR, CDI and JPA Sample: Online Polling
sample : mongoDBTestClientSample : MongoDB Test Client Sample
opensource : cassandraDBSample : Cassandra Sample
opensource : StripesSample : Stripes v1.5.7 Integration
opensource : Struts1Sample : Struts v1.3.8 Integration

5. 开发调试环境搭建

Eclipse 最初是由 IBM 公司开发的替代商业软件 Visual Age for Java 的下一代 IDE 开发环境。

2001 年 11 月贡献给开源社区，现在它由非营利软件供应商联盟 Eclipse 基金会（Eclipse Foundation）管理。

2003 年，Eclipse 3.0 选择 OSGi 服务平台规范为运行时架构。

2007 年 6 月，稳定版 3.3 发布；

2008 年 6 月发布代号为 Ganymede 的 3.4 版；

2009 年 6 月发布代号为 Galileo 的 3.5 版；

2010 年 6 月发布代号为 Helios 的 3.6 版；

2011 年 6 月发布代号为 Indigo 的 3.7 版；

2012 年 6 月发布代号为 Juno 的 4.2 版；

2013 年 6 月发布代号为 Kepler 的 4.3 版；

2014 年 6 月发布代号为 Luna 的 4.4 版；

2015 年 6 月项目要发布代号为 Mars 的 4.5 版。

基于 Eclipse 插件开发环境（Plug-in Development Environment，PDE）所提供的插件技术，开发者可以构建与 Eclipse 环境无缝集成的工具。

同样，IBM 的 Liberty 开发环境也是基于这之上进行插件开发并增强的，接下来介绍 Liberty 在 Eclipse 之上的开发环境搭建。

首先，下载 Eclipse 4.4.2 for Java EE Developers (Luna SR2)

<https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/lunasr2>

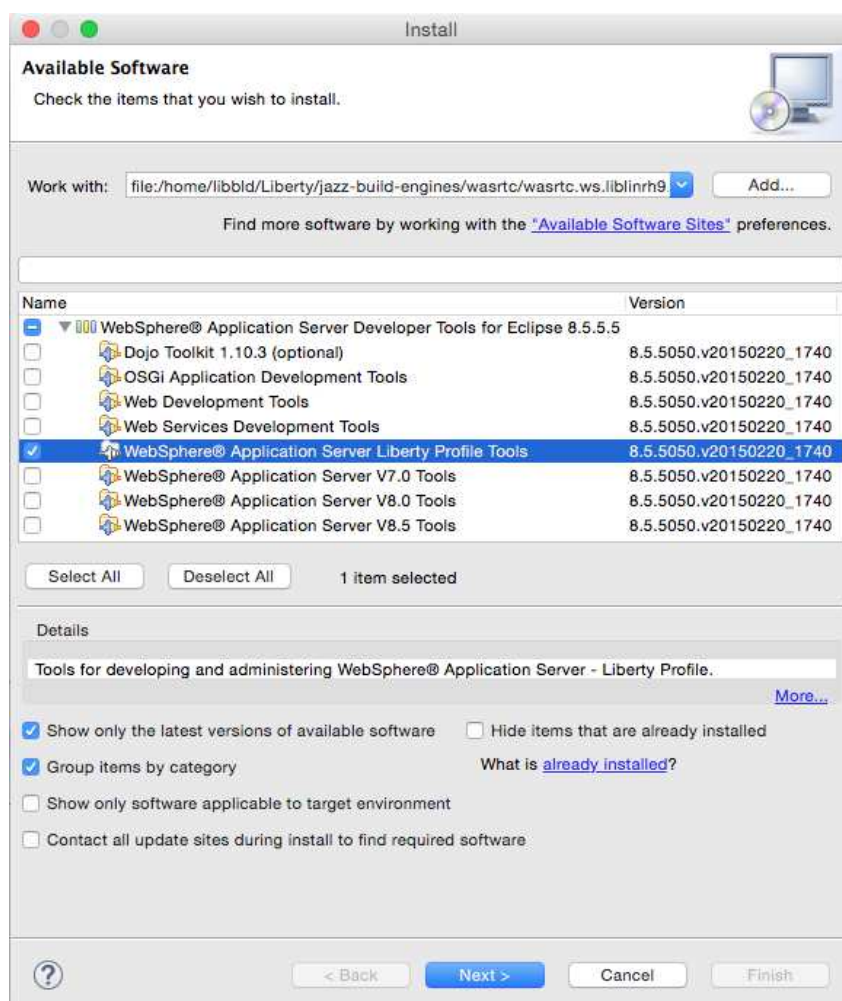
下载 WDT（WebSphere Developer Tools）

https://developer.ibm.com/wasdev/downloads/liberty-profile-using-eclipse/wdt/luna/wdt-update-site_8.5.5.5.v20150220_1740.zip

提示：实际上，根据 Eclipse 开发工具的更新与开发进度可以下载到更新或更加稳定的版本，对应下载即可。

5.1. 本地文件安装

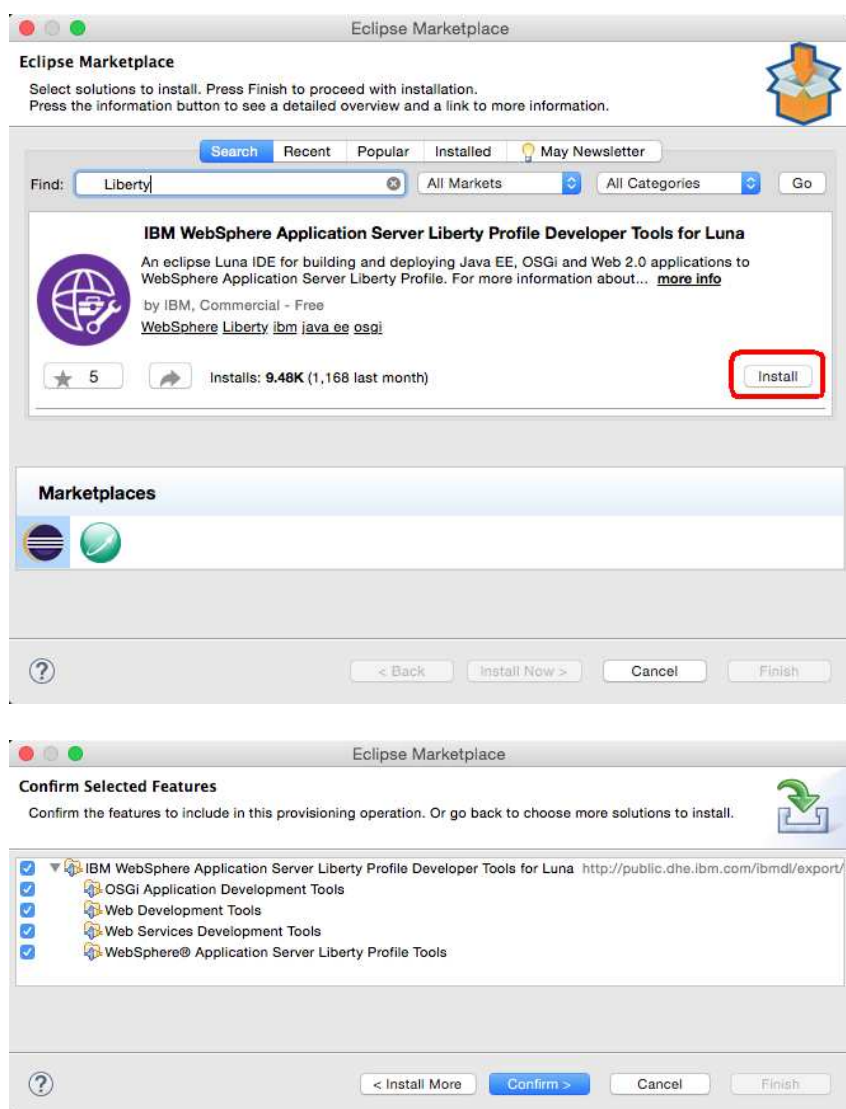
将下载的 Eclipse 包解压之后, 然后启动 Eclipse; 再通过菜单“Help” → “Install New Software...”, 然后“Add...” → “Archive...”, 选择下载的 WDT 包, 这样就可以直接通过本地包的方式安装 WDT 了, 如下图所示:

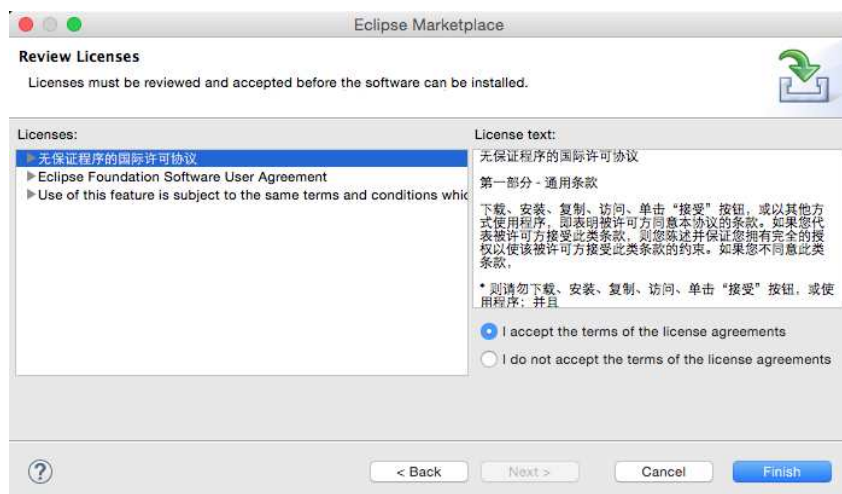
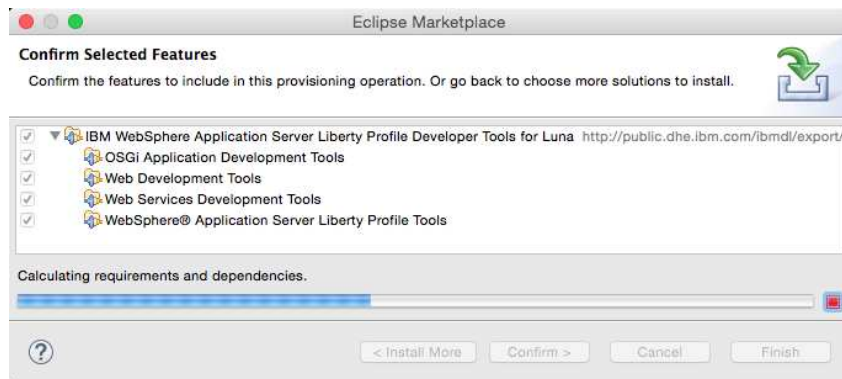


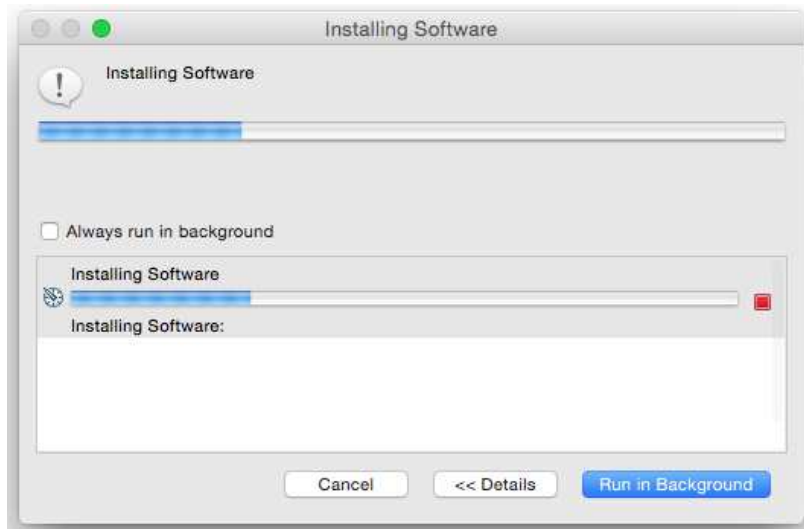
安装完成之后, 重新启动 Eclipse 即可。

5.2. 在线安装

另一种可以直接通过在线的方式进行安装，将下载的 Eclipse 包解压之后，然后启动 Eclipse；再通过菜单“Help” → “Eclipse Marketplace...”，输入“Liberty”进行查找，如下图所示，然后点击“Install”进行安装：





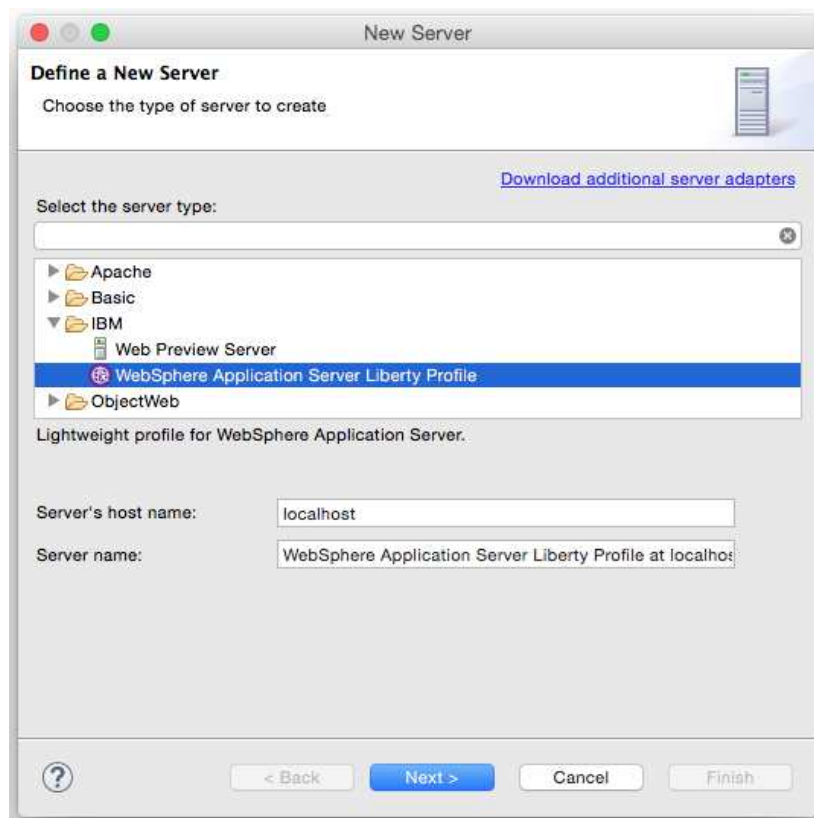


根据网络的状况决定下载与安装的时间,如果网络状况比较好,可以直接进行在线安装,但是仍会花不少时间,对于要构建多个开发环境建议通过本地文件包方式进行安装,这更加适合多人开发时进行快速安装。

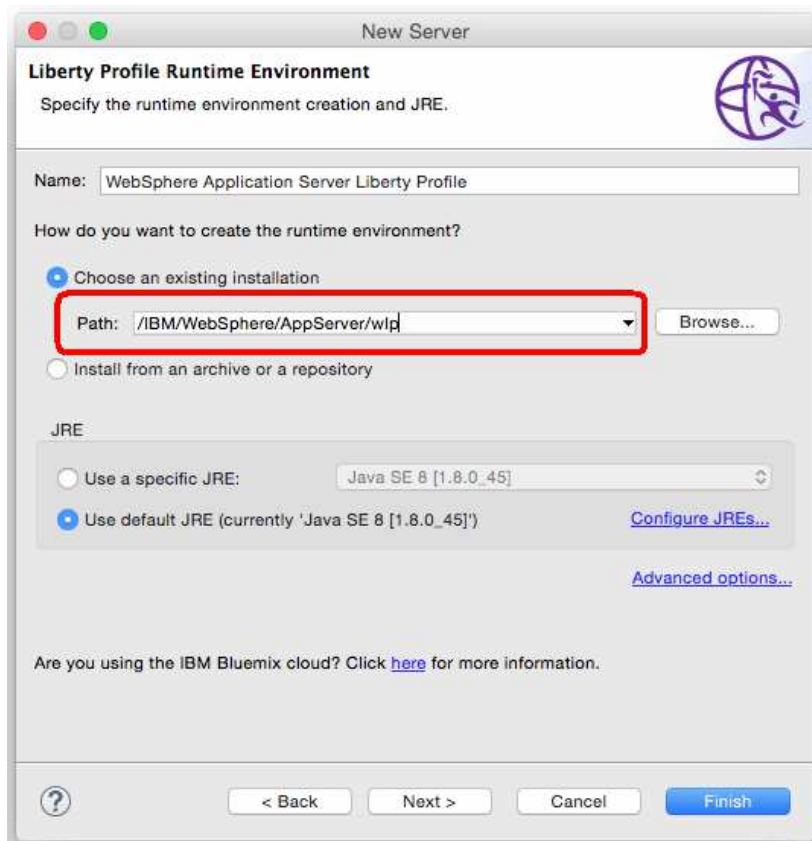
6. Eclipse 中配置使用 Liberty 服务器

6.1. 配置 Liberty 服务器

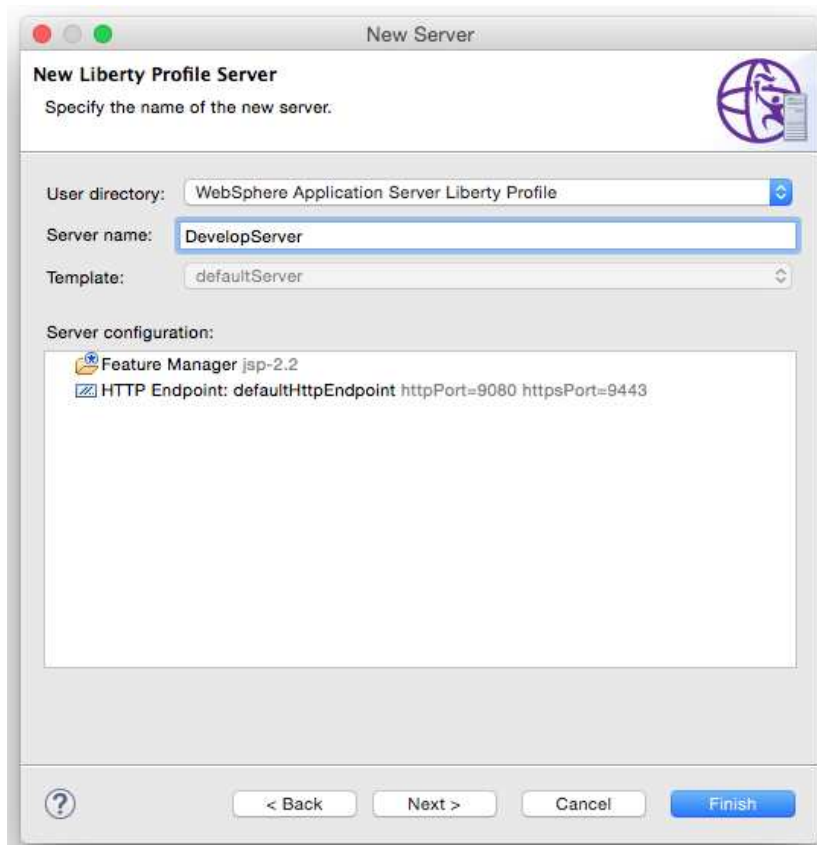
安装结束后，重新启动 Eclipse 之后。再通过 “Servers” 视图，右键，“New”，就可以启动创建新的服务器的向导，来创建 “WebSphere Application Server Liberty Profile” 了，在配置过程中指向之前 Liberty 所解压的路径。



输入 Liberty 解压后的全路径，如下图所示：



输入服务器名称，比如“DevelopServer”，如下图所示：

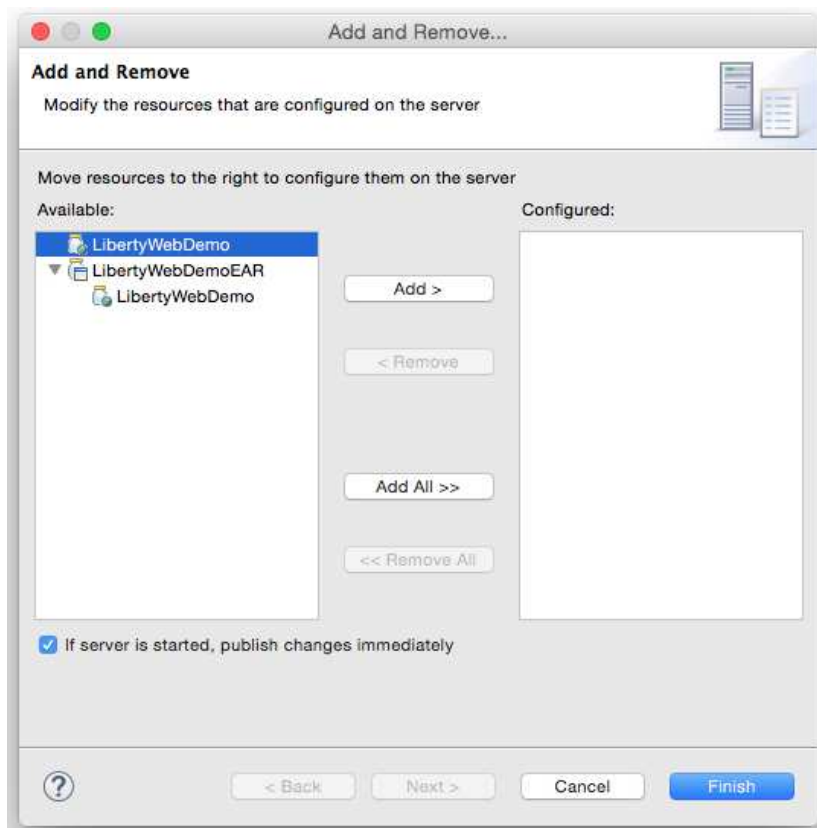


这样，就可以直接创建出来的新服务器 Liberty Profile 了。



6.2. 创建 Web 项目并调试

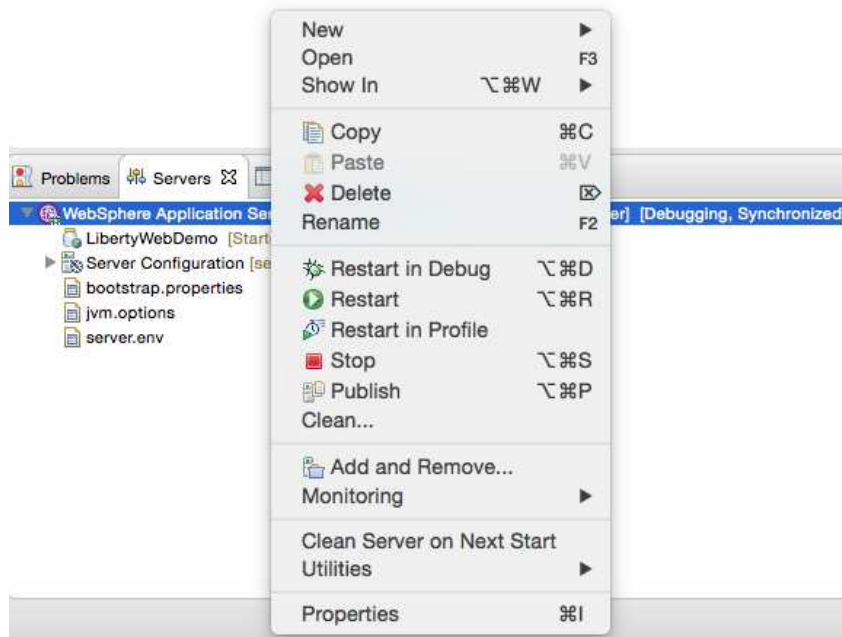
新创建 Web 项目，并将项目增加或移出服务器，如下图所示：



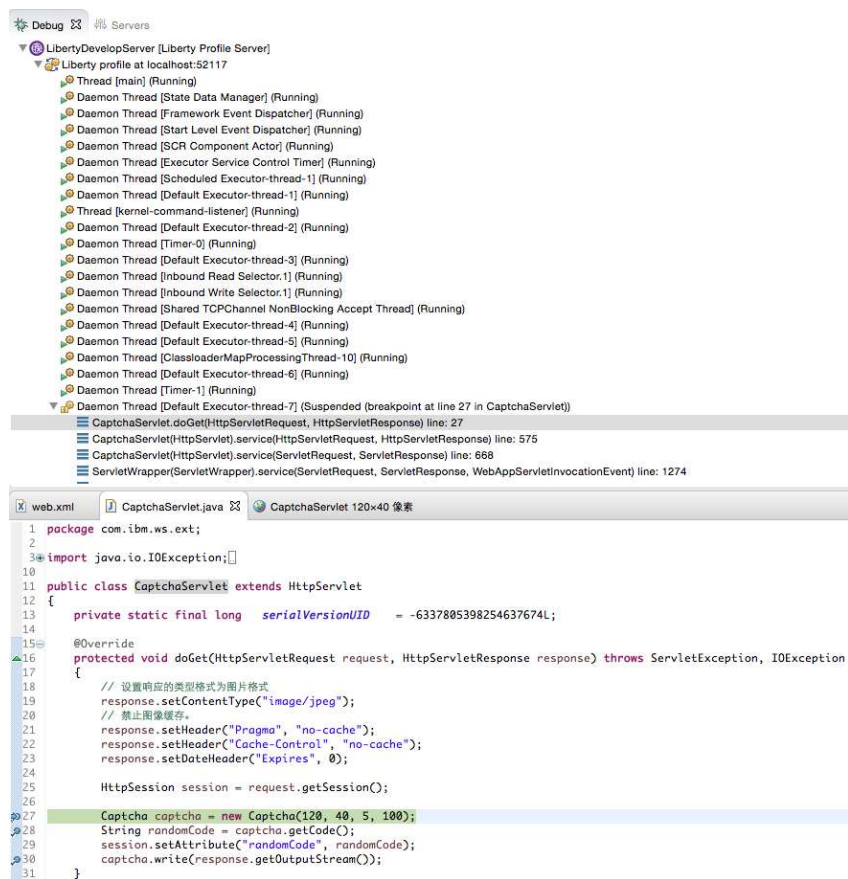
在应用程序开发视图中设置断点进行跟踪调试，如下图所示：

```
3 import java.io.IOException;
10
11 public class CaptchaServlet extends HttpServlet
12 {
13     private static final long serialVersionUID = -6337805398254637674L;
14
15     @Override
16     protected void doGet(HttpServletRequest request, HttpServletResponse response)
17     {
18         // 设置响应的类型格式为图片格式
19         response.setContentType("image/jpeg");
20         // 禁止图像缓存。
21         response.setHeader("Pragma", "no-cache");
22         response.setHeader("Cache-Control", "no-cache");
23         response.setDateHeader("Expires", 0);
24
25         HttpSession session = request.getSession();
26
27         Captcha captcha = new Captcha(120, 40, 5, 100);
28         String randomCode = captcha.getCode();
29         session.setAttribute("randomCode", randomCode);
30         captcha.write(response.getOutputStream());
31     }
32 }
```

然后，以调试模式启动 Liberty 服务器，也有其他更多的操作菜单，重启、停止等等。



打开浏览器访问 Web 应用程序，在访问到断点的代码时，就可以切入到代码视图中，进行调试跟踪模式了，如下图所示：



代码调试跟踪的其他种种操作在此则省略不进行详细介绍，在这里进行代码的修改然后再调试则相当的快速与方便了。

6.3. Maven

Liberty 提供有 Maven 插件来进行自动化部署应用与服务器的启停等操作。

6.3.1. 插件仓库

为了使用这个功能，需要在项目的 `pom.xml` 文件中增加 Liberty 插件，公共的仓库如下：

```
<pluginRepository>
  <id>Liberty</id>
  <name>Liberty Repository</name>
  <url>http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/maven/re
pository/</url>
  <layout>default</layout>
  <snapshots>
    <enabled>>false</enabled>
  </snapshots>
  <releases>
    <enabled>>true</enabled>
  </releases>
</pluginRepository>
```

以下的配置进行加载 Liberty 的 Maven 插件，并指定了基础的配置信息。注意 `serverHome` 与 `serverName` 更改为自己对应的路径与名称。

```
<build>
  <plugins>
    <!-- Enable liberty-maven-plugin -->
    <plugin>
      <groupId>com.ibm.websphere.wlp.maven.plugins</groupId>
      <artifactId>liberty-maven-plugin</artifactId>
      <version>1.0</version>
      <!-- Specify configuration -->
      <configuration>
        <serverHome>/IBM/WebSphere/AppServer/wlp</serverHome>
        <serverName>DevelopServer</serverName>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
</plugin>
</plugins>
</build>
```

接下来就可以进行更加具体的操作了：

6.3.2. 服务器的管理

创建服务器

```
mvn liberty:create-server
```

启动服务器

```
mvn liberty:start-server
```

停止服务器

```
mvn liberty:stop-server
```

检查服务器状态

```
mvn liberty:status-server
```

打包服务器

```
mvn liberty:package-server -DpackageFile=packaged.zip
```

从打包文件中安装服务器

```
mvn liberty:install-server -DassemblyArchive=packaged.zip
```

6.3.3. 部署或删除应用程序

```
mvn liberty:deploy -DappArchive=Java2Class.ear
```

```
mvn liberty:undeploy -DappArchive=Java2Class.ear
```

6.3.4. Maven 私服搭建

<http://www.sonatype.org/nexus/thank-you-for-downloading/?dl=zip>

请参见以下链接：

6.4. ANT

Liberty 提供有 ANT 插件来进行自动化部署应用与服务器的启停等操作。

插件 wlp-anttasks.jar 位于 dev/tools/ant 目录下, 拷贝到 ANT 的 lib 目录下或者指定 CLASSPATH 均可以。

然后在 ANT 的项目文档中通过 antlib 声明

```
<project .... xmlns:wlp="antlib:com.ibm.websphere.wlp.ant">
...
</project>
```

6.4.1. 服务器的管理

创建服务器

```
<wlp:server installDir="{wlp_install_dir}" operation="create"/>
```

启动服务器

```
<wlp:server installDir="{wlp_install_dir}" operation="start" serverName="{serverName}" />
```

停止服务器

```
<wlp:server installDir="{wlp_install_dir}" operation="stop" serverName="{serverName}" />
```

检查服务器状态

```
<wlp:server installDir="{wlp_install_dir}" operation="status"/>
```

打包服务器

```
<wlp:server installDir="{wlp_install_dir}" operation="package" archive="packaged.zip"/>
```

6.4.2. 部署或删除应用程序

```
<wlp:deploy file="{basedir}/resources/Java2Class.ear" timeout="40000"/>
```

```
<wlp:undeploy file="Java2Class.ear" timeout="60000" />
```

7. Liberty 应用开发

应用程序开发的内容较多，这里暂时仅列出一些应用程序开的相关列表，待后面进行慢慢补充具体的开发内容。

7.1. 简单应用程序开发

7.1.1. JSP、Servlet 应用程序开发

将介绍最简单的 JSP、Servlet 应用程序开发。

7.1.2. JAX-RS RESTful Web Services 应用程序开发

将介绍 REST Web Services 相关的开发技术。

7.1.3. CDI 应用程序开发

Context and Dependency Injection (CDI)

7.1.4. JAX-WS Web Services 应用程序开发

将介绍 REST Web Services 相关的开发技术。

7.1.5. WebSocket 应用程序开发

将介绍 WebSocket 相关的开发技术。

7.2. OSGi 应用程序开发

将介绍 OSGi 应用程序相关的开发技术。

7.3. 企业应用程序开发

7.3.1. 数据库访问应用程序开发

无数据不应用，将介绍应用程序与数据库交互操作的相关开发技术。

7.3.2. EJB 应用程序开发

无。

7.3.3. JMS 应用程序开发

将介绍 JMS 消息的相关开发技术。

7.3.4. 异步、并发应用程序开发

将介绍如何进行异步、并发应用程序开发的相关内容。

7.3.5. JavaMail 邮件应用程序开发

将介绍与 JavaMail 邮件的相关开发技术。

8. 管理控制台安装使用

8.1. 安装 adminCenter 组件

由于默认并没有直接带有管理控制台，需要附加安装，关于 adminCenter 组件的详细信息可以查看

<https://developer.ibm.com/wasdev/downloads/#asset/features-com.ibm.websphere.appserver.adminCenter-1.0>

接着执行如下命令进行安装所需要的组件：

- 查找 adminCenter 组件

installUtility find admin

```
正在建立与已配置存储库的连接...
此过程可能要花几分钟完成。

已成功连接至所有已配置的存储库。

正在搜索资产。此过程可能要花几分钟完成。

feature : adminCenter-1.0 : Admin Center
feature : localConnector-1.0 : Admin Local Connector
feature : restConnector-1.0 : Admin REST Connector 1.0
feature : restConnector-2.0 : Admin REST Connector 2.0
sample : OnlinePollingSample : EJB in WAR, CDI and JPA Sample: Online Polling
```

可以看到列表中包含有 adminCenter-1.0 组件，于是接下来可以安装之。

- 安装查找 adminCenter 组件

installUtility install adminCenter-1.0

```
命令提示符 - installUtility install adminCenter-1.0
C:\IBM\WebSphere\liberty\18.0.0.2\wlp-webProfile8-java8-win-x86_64-18.0.0.2\bin>installUtility install adminCenter-1.0
正在建立与已配置存储库的连接...
此过程可能要花几分钟完成。

已成功连接至所有已配置的存储库。

准备安装资产。此过程可能要花几分钟完成。

Eclipse Public License - v 1.0
THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE
PUBLIC
LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE
PROGRAM
CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

选择 "[1] 我同意"，或 "[2] 我不同意"： 1
```

输入“1”来接受 Liberty 许可协议 Eclipse Public License – v 1.0，然后开始下载安装，如下图所示：

```
选择“[1] 我同意”，或“[2] 我不同意”： 1

第 1 个步骤 (共 18 个步骤)：正在下载 servlet-3.0...
第 2 个步骤 (共 18 个步骤)：正在安装 servlet-3.0...
第 3 个步骤 (共 18 个步骤)：正在下载 servlet-3.1...
第 4 个步骤 (共 18 个步骤)：正在安装 servlet-3.1...
第 5 个步骤 (共 18 个步骤)：正在下载 jaxrsClient-2.0...
第 6 个步骤 (共 18 个步骤)：正在安装 jaxrsClient-2.0...
第 7 个步骤 (共 18 个步骤)：正在下载 restConnector-1.0...
第 8 个步骤 (共 18 个步骤)：正在安装 restConnector-1.0...
第 9 个步骤 (共 18 个步骤)：正在下载 jaxrs-2.0...
第 10 个步骤 (共 18 个步骤)：正在安装 jaxrs-2.0...
第 11 个步骤 (共 18 个步骤)：正在下载 jaxrs-1.1...
第 12 个步骤 (共 18 个步骤)：正在安装 jaxrs-1.1...
第 13 个步骤 (共 18 个步骤)：正在下载 jsp-2.2...
第 14 个步骤 (共 18 个步骤)：正在安装 jsp-2.2...
第 15 个步骤 (共 18 个步骤)：正在下载 adminCenter-1.0...
第 16 个步骤 (共 18 个步骤)：正在安装 adminCenter-1.0...
第 17 个步骤 (共 18 个步骤)：正在验证安装的修订 ...
第 18 个步骤 (共 18 个步骤)：正在清除临时文件...

已成功安装所有资产。

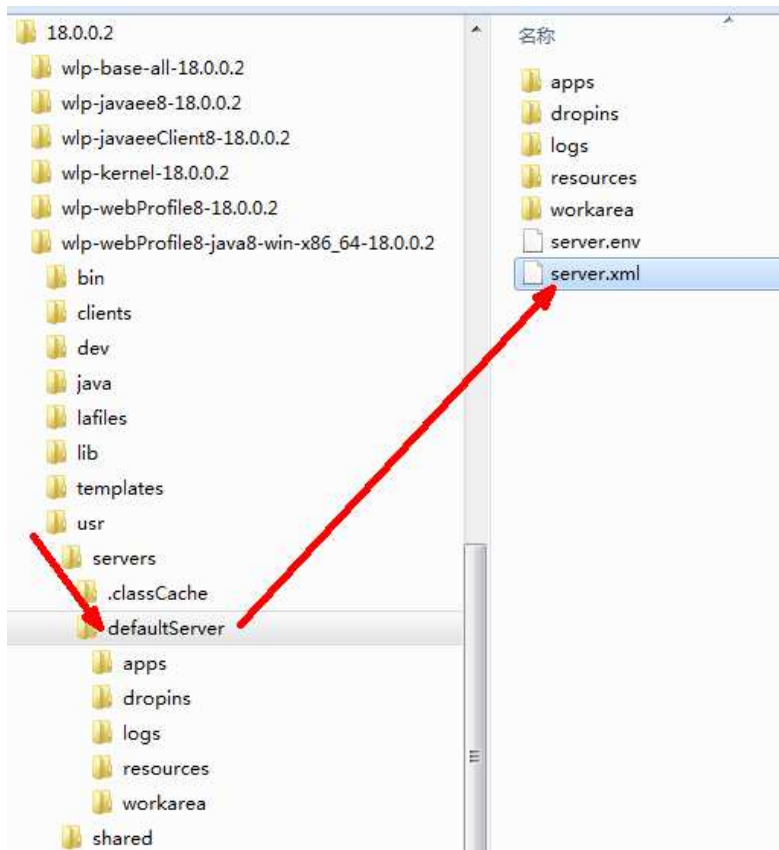
启动产品验证...
产品验证已成功完成。
```

根据网络的速度情况，经过一段时间后可以顺利安装完成。通过上图可以知道一共安装如下 9 个组件：

- adminCenter-1.0 [1.0.0]
- distributedMap-1.0 [1.0.0]
- jaxrs-1.1 [1.0.0]
- jndi-1.0 [1.0.0]
- json-1.0 [1.0.0]
- jsp-2.2 [1.0.0]
- restConnector-1.0 [1.0.0]
- servlet-3.0 [1.0.0]
- ssl-1.0 [1.0.0]

8.2. 添加 adminCenter 配置

在确认安装完成后，在 `server.xml` 中增加如下控制台的配置信息



编辑 server.xml 配置文件，将以下配置信息，直接拷贝进去附加到最后即可。

```
<featureManager>
  <feature>adminCenter-1.0</feature>
  <feature>ssl-1.0</feature>
</featureManager>

<administrator-role>
  <user>admin</user>
</administrator-role>

<basicRegistry id="basic">
  <user
    name="admin"
    password="{aes}APBXmPlx9ilC3j3MTHEC0u8i2M1VLCuvwxj3gFUXMIhu"/>
  </basicRegistry>

  <keyStore
    id="defaultKeyStore"
    password="{aes}APBXmPlx9ilC3j3MTHEC0u8i2M1VLCuvwxj3gFUXMIhu" />
```


8.3. 启动 adminCenter 服务

- 启动默认的 defaultServer 服务器

server start

如果之前启动了 defaultServer 服务器，那么配置信息添加进去后，服务器会自动加载这些配置，不需要重新启动它即可启动 adminCenter 服务了。

可以在日志里看到如下信息：

[AUDIT] CWWKG0017I: 服务器配置已成功地在 2.771 秒内更新。

[AUDIT] CWWKF0012I: 服务器已安装下列功能部件：[restConnector-2.0, json-1.0, adminCenter-1.0]。

[AUDIT] CWWKF0008I: 已在 2.608 秒内完成功能部件更新。

[AUDIT] CWWKT0016I: Web 应用程序可用 (default_host) :
http://localhost:9080/ibm/adminCenter/serverConfig-1.0/

[AUDIT] CWWKT0016I: Web 应用程序可用 (default_host) :
http://localhost:9080/ibm/api/

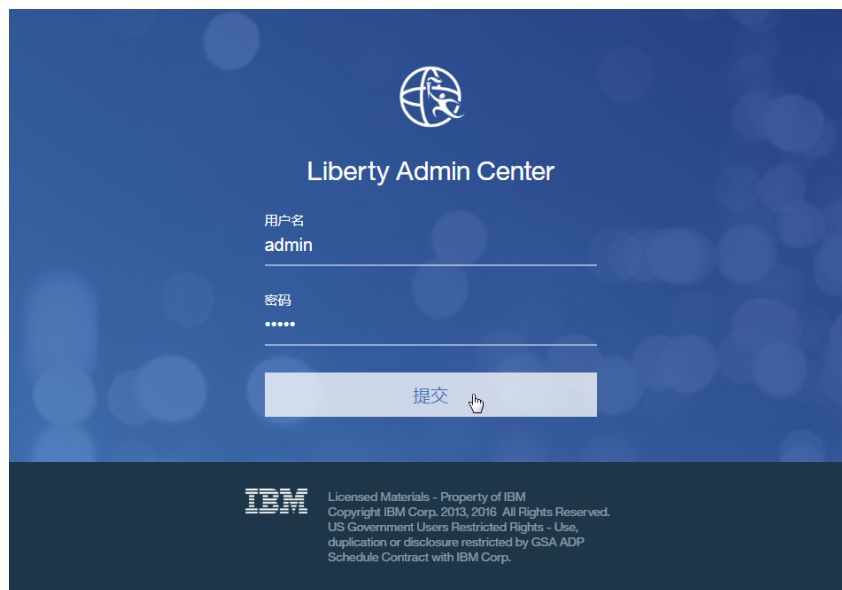
[AUDIT] CWWKT0016I: Web 应用程序可用 (default_host) :
http://localhost:9080/IBMJMXConnectorREST/

[AUDIT] CWWKT0016I: Web 应用程序可用 (default_host) :
http://localhost:9080/ibm/adminCenter/explore-1.0/

[AUDIT] CWWKT0016I: Web 应用程序可用 (default_host) :
http://localhost:9080/adminCenter/

然后通过浏览器访问 <http://localhost:9080/adminCenter/> 就可以访问控制台了,由于配置了 https, 因此会直接跳转到 <https://localhost:9443/adminCenter/> , 因此可以直接访问后面的地址即可, 登录用户名为 admin , 密码为 P@ssw0rd

登录界面如下图所示:



Liberty Admin Center

用户名
admin

密码
.....

提交

IBM Licensed Materials - Property of IBM
Copyright IBM Corp. 2013, 2016 All Rights Reserved.
US Government Users Restricted Rights - Use,
duplication or disclosure restricted by GSA ADP
Schedule Contract with IBM Corp.

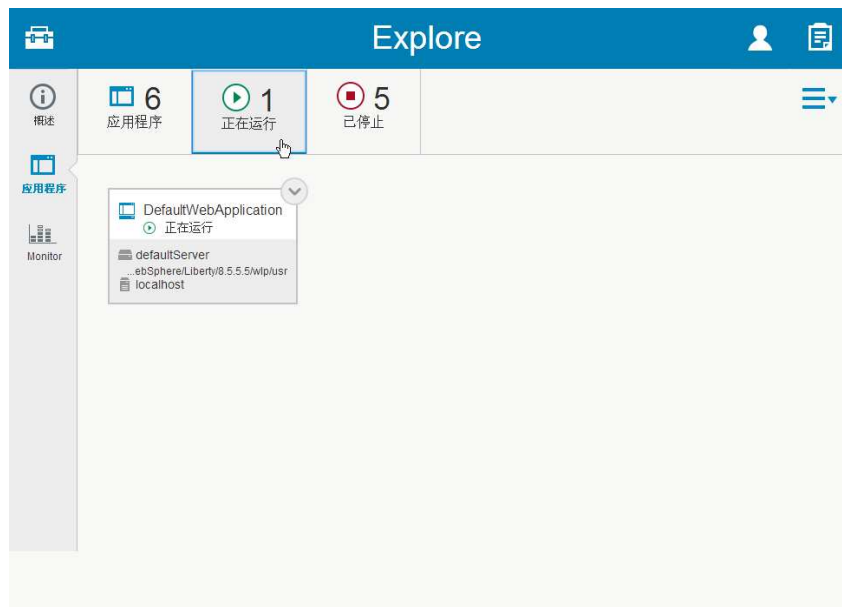
输入设置的用户名和密码后，登录进管理控制台，如下图所示：



由于我们仅仅是配置 adminCenter 功能而已，因此仅有以上比较简单的两个链接按钮。

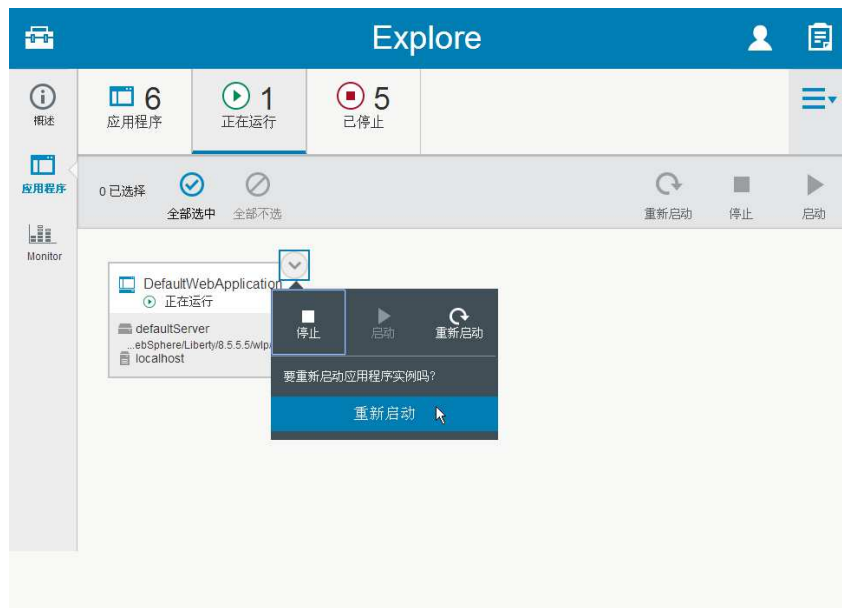
8.4. 使用管理控制台

通过 “Explore” 链接进入应用与性能监控页面，如下图所示：



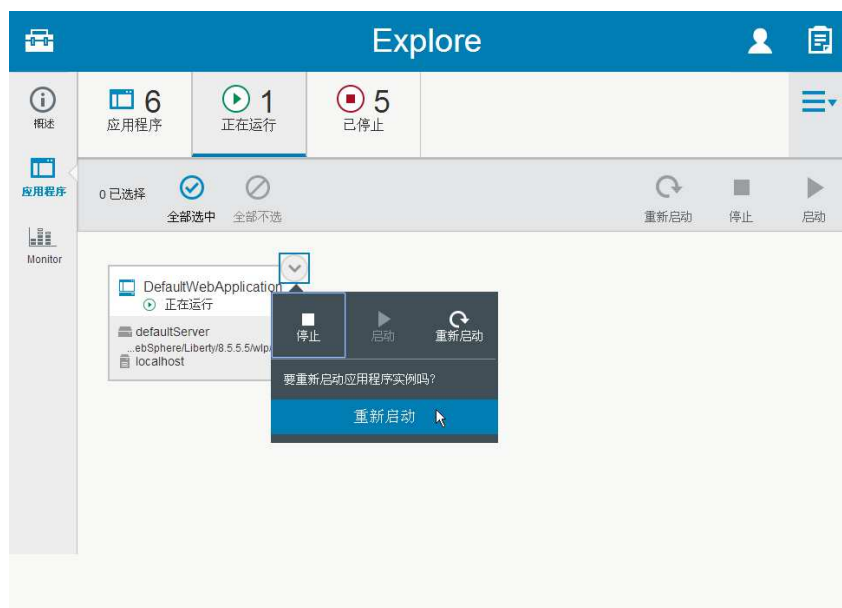
8.5. 操作应用程序

可以对应用程序进行“停止”、“启动”、“重新启动”等操作，如下图所示：



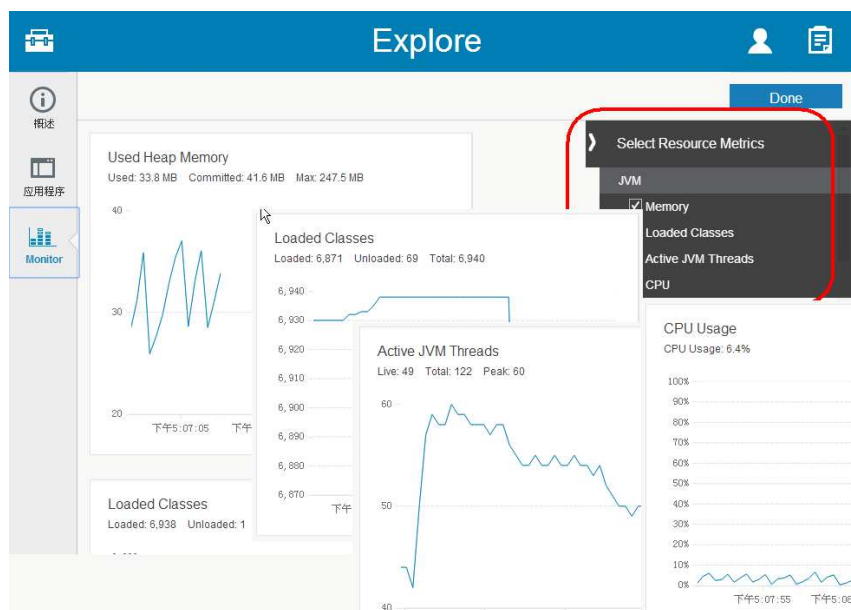
8.6. 操作服务器

可以在管理控制台将服务器进行停止，但是服务器停止之后，就得需要从命令行进行再启动了，无法通过管理控制台进行启动的。如下图所示：



8.7. 查看 JVM 信息

可以在管理控制台查看 JVM 的相关信息，比如堆大小的利用情况、加载的类的数量、活动的线程数以及 CPU 的利用率情况。如下图所示：



关于更多的管理控制台使用细节，请参见后续章节中的集群配置与管理内容。

9. 生产环境配置

9.1. JDK 配置

9.1.1. 系统 JAVA_HOME 配置 JDK

在 Windows 下:

```
set JAVA_HOME=C:\IBM\jre6  
set PATH=%JAVA_HOME%\bin;%PATH%
```

Linux 下

```
export JAVA_HOME=/opt/IBM/jre6  
export PATH=$JAVA_HOME:$PATH
```

Liberty 通过 JAVA_HOME、JRE_HOME、PATH 这三个环境变量进行搜索 java 命令

9.1.2. 通过 server.env 配置 JDK

```
${wlp.install.dir}/etc/server.env
```

对所有的 server 起作用

```
${server.config.dir}/server.env
```

只对单个 server 起作用

然后在 server.env 文件中进行键值对的配置即可，如下：

```
JAVA_HOME=/opt/IBM/jre6
```

9.1.3. 通过 jvm.options 配置 JVM 参数

```
${wlp.install.dir}/etc/jvm.options
```

对所有的 server 起作用

```
${server.config.dir}/jvm.options
```

只对单个 server 起作用

比如，配置 -X 特殊的 JVM 参数，或者 -D 之类的变量设置。

9.2. 共享库配置

在 server.xml 文件中增加

```
<classloader >
  <commonLibrary>
    <fileset includes="utility.jar"/>
  </commonLibrary>
</classloader>
```

这是整个服务器共有的加载库

如果仅对某些应用进行配置，则可以通过如下配置来达成

```
<library id="commons">
  <fileset dir="{server.config.dir}/logger" includes="commons.jar"/>
</library>
<application name="App1" location="App1.ear">
  <classloader commonLibraryRef="commons"/>
</application>
<application name="App2" location="App2.ear">
  <classloader commonLibraryRef="commons"/>
</application>
```

全局共享库

可以直接将相关的文件拷贝到如下目录即可。

shared/config/lib/global

servers/server_name/lib/global

这样所有的应用共享这些类库了。

但是如果你对某个应用配置了 classloader，并且要使用这个全局共享库，则需要配置

```
<application name="App3" location="App3.ear">
  <classloader commonLibraryRef="global"/>
</application>
```

9.3. 安全配置

通过启用 SSL 通讯，在客户端与 Liberty 服务器之间启用安全通道 HTTPS 进行数据传输。进而，也配置数字证书进行安全传输与身份识别等安全配置。

9.3.1. 生成证书

通过密钥和证书管理工具 `keytool` 命令来生成证书：

```
set JAVA_HOME= C:\IBM\WebSphere\AppServer\9.0\java\8.0
```

```
cd /d %JAVA_HOME%\bin
```

RSA 密钥对可以采用 **512-2048** 位，**DSA** 密钥对可以采用 **512-2048** 位，均需为 64 倍数位。

```
%JAVA_HOME%\bin\keytool -genkey -alias Liberty -keyalg RSA -keysize 4096 -dname  
CN=IBMChina -keystore C:\IBM\IBMChina.jks -storepass Liberty -keypass Liberty -validity  
3650 -v
```

```
%JAVA_HOME%\bin\keytool -genkey -alias Liberty -keyalg DSA -keysize 2048 -dname  
CN=IBMChina -keystore C:\IBM\IBMChina.jks -storepass Liberty -keypass Liberty -validity  
3650 -v  
dir C:\IBM\IBMChina.jks
```

密钥和证书管理工具命令：

<code>-certreq</code>	生成证书请求
<code>-changealias</code>	更改条目的别名
<code>-delete</code>	删除条目
<code>-exportcert</code>	导出证书
<code>-exportseckey</code>	批量导出密钥
<code>-genkeypair</code>	生成密钥对
<code>-genseckey</code>	生成密钥
<code>-gencert</code>	根据证书请求生成证书
<code>-importcert</code>	导入证书或证书链
<code>-importpass</code>	导入密码
<code>-importkeystore</code>	从另一个密钥库导入一个或所有条目
<code>-importseckey</code>	批量导入密钥
<code>-keypasswd</code>	更改条目的密钥密码
<code>-list</code>	列出密钥库中的条目

-printcert	打印证书的内容
-printcertreq	打印证书请求的内容
-printcrl	打印 CRL 文件的内容
-storepasswd	更改密钥库的库密码

使用 “keytool -command_name -help” 以了解 command_name 的用法

keytool -genkeypair [OPTION]...

生成密钥对选项:

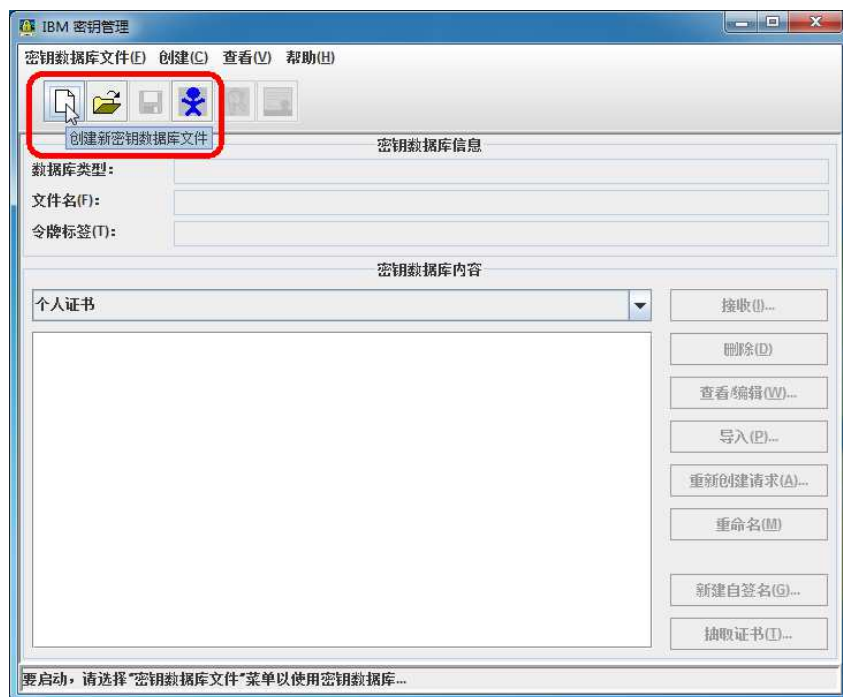
-alias <alias>	要处理的条目的别名
-keyalg <keyalg>	密钥算法名称
-keysize <keysize>	密钥位大小
-sigalg <sigalg>	签名算法名称
-dname <dname>	专有名称
-startdate <startdate>	证书生效日期/时间
-ext <value>	X.509 扩展
-validity <valDays>	有效期天数
-keypass <arg>	密钥密码
-keystore <keystore>	密钥库名称
-storepass <arg>	密钥库密码
-storetype <storetype>	密钥库类型
-providername <providername>	提供程序名
-providerclass <providerclass>	提供程序类名
-providerarg <arg>	提供程序自变量
-providerpath <pathlist>	提供程序类路径
-v	详细输出
-protected	采用保护机制的密码

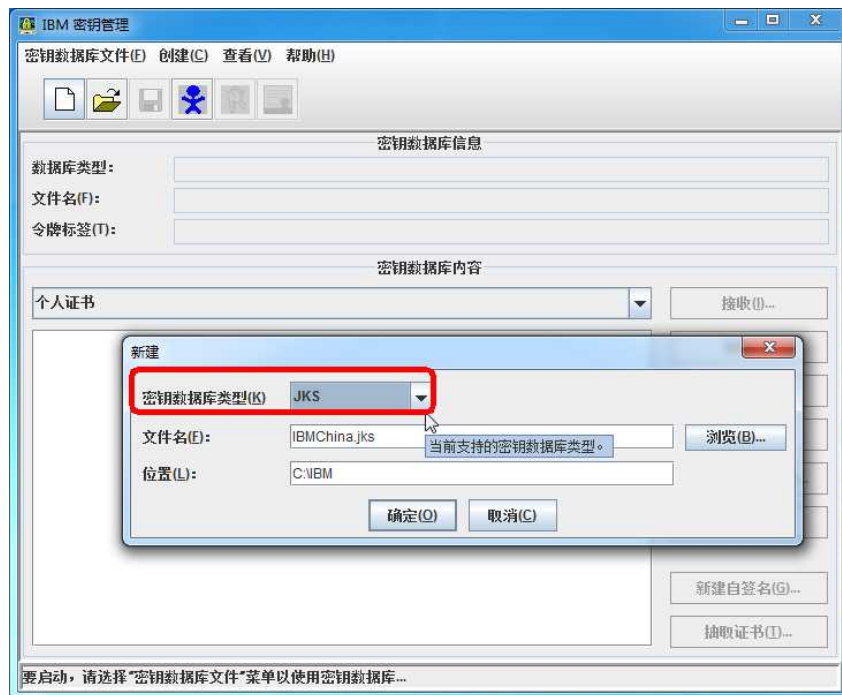
除了通过 keytool 命令行的密钥和证书管理工具来进行证书管理之外，也可以通过 IBM WAS 自带的 ikeyman 图形化的证书管理工作来进行管理，更加便捷与高效。

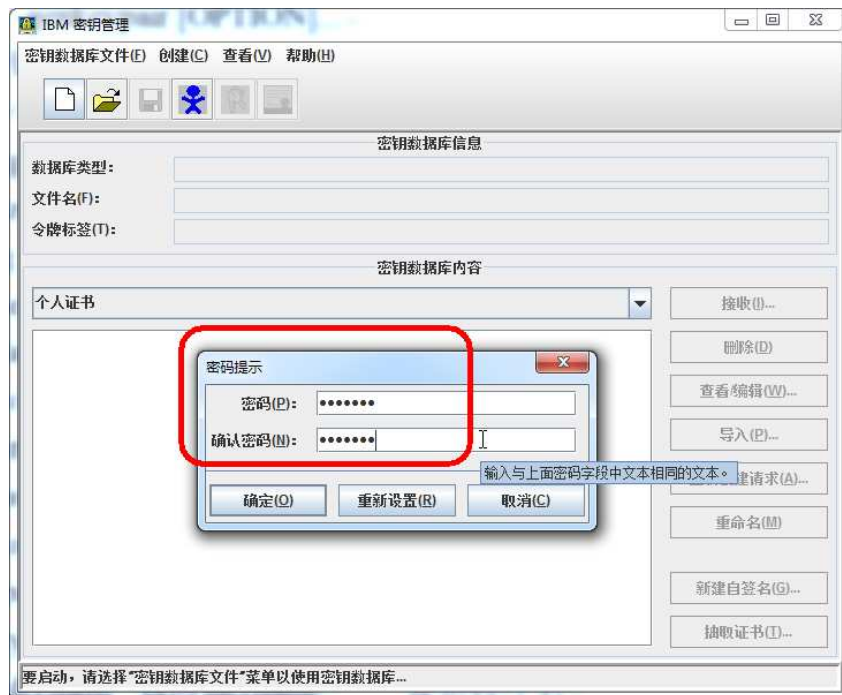
```
set WAS_HOME= C:\IBM\WebSphere\AppServer\9.0
```

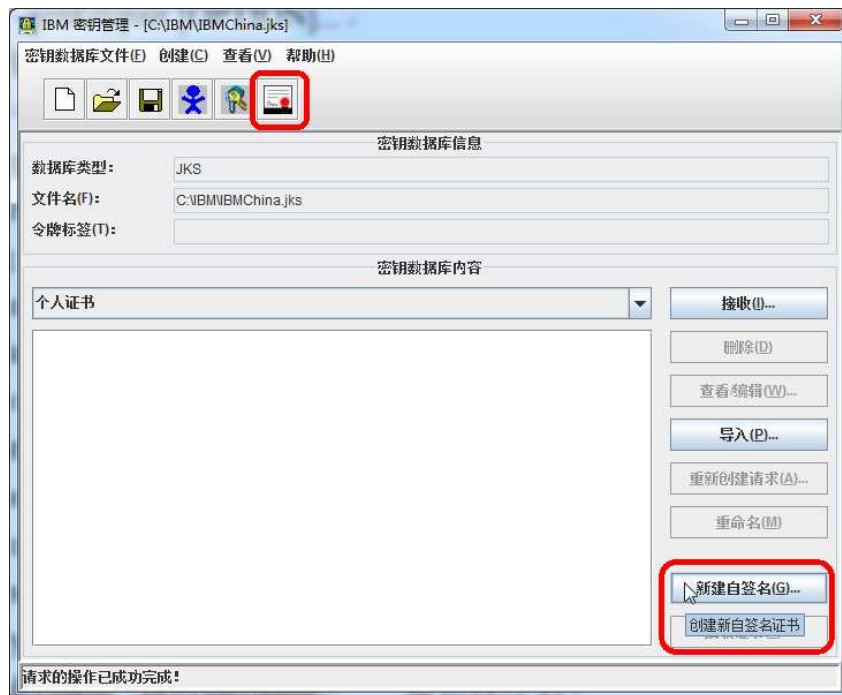
```
cd /d %WAS_HOME%\bin
```

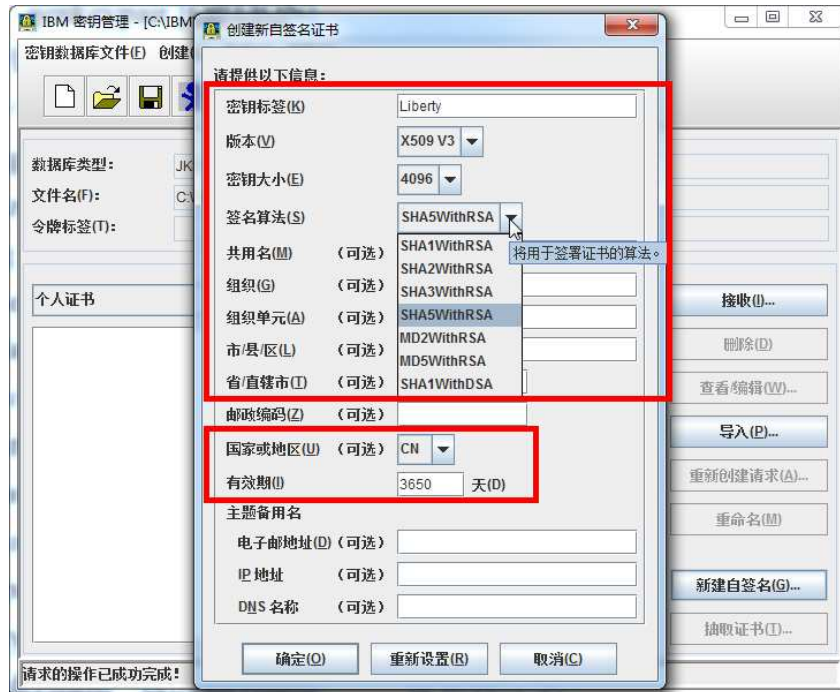
```
%WAS_HOME%\bin\ikeyman
```











9.3.2. 配置 Liberty

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="SslServer">
  <featureManager>
    <feature>jsp-2.3</feature>
    <feature>ssl-1.0</feature>
  </featureManager>

  <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="9443" />
  <!-- The password is Liberty -->
  <keyStore id="Liberty_SSL_KeyStore" location="C:\IBM\IBMChina.jks"
type="JKS" password="{xor}EzY9Oi0rJg==" />
  <ssl id="Liberty_SSL_Settings" keyStoreRef="Liberty_SSL_KeyStore"
clientAuthenticationSupported="true" />
  <sslDefault sslRef="Liberty_SSL_Settings" />
</server>
```

其中 `sslRef="Liberty_SSL_Settings"` 默认值为 `<sslDefault sslRef="defaultSSLConfig" />`
`{xor}EzY9Oi0rJg==` 这个密码可以是明文的 Liberty 密码，但是显然不合适：

可以通过 Liberty 下 bin 目录下的加密工具 securityUtility 进行加密，比如

```
securityUtility encode Liberty
```

得到加密后的密码 `{xor}EzY9Oi0rJg==`

加密工具 securityUtility 详细用法：

```
securityUtility encode [选项]
```

描述：

编码提供的文本。

选项：

```
--encoding=[xor|aes|hash]
```

指定如何对密码进行编码。受支持的编码为

xor、aes 和 hash。缺省编码为 xor。

```
--key=[key]
```

指定使用 AES 进行编码时要使用的密钥。将对此字符串

使用散列算法，以生成将用于对密码加密和解密的加密密钥。

可以通过定义变量 `wlp.password.encryption.key`

（该变量的值为密钥）来将密钥提供给服务器。

如果未提供此选项，那么将使用缺省密钥。

--listCustom

以 JavaScript 对象表示法 (JSON) 格式显示定制密码加密的信息。

此信息包含：

name: 定制密码加密算法名称

featurename: 功能部件名称

description: 定制密码加密的描述

--notrim

指定是否要从所指定文本的开头和末尾除去空格字符。

如果指定了此选项，那么所提供的文本将按原样编码。

如果未指定此选项，那么将除去所指定文本的开头和末尾的空格字符。

textToEncode

如果未指定参数，此工具将输入交互方式：

否则，将对所提供文字进行编码。

如果将带空格的文本指定为参数，那么必须用引号将其完全引起来。

缺省编码为 **xor**，可以不指定编码

```
securityUtility encode --encoding=xor Liberty
```

```
securityUtility encode Liberty
```

aes 编码

```
securityUtility encode --encoding=aes --key=Liberty Liberty
```

hash 编码

```
securityUtility encode --encoding=hash Liberty
```

9.3.3. 启动 Liberty 加载新配置

启动 Liberty 即可，如果在配置之前已启动，则会自动重新进行了加载，通过日志文件检查配置信息是否成功加载，并通过浏览器确认能够正确地通过 https 进行网页访问。

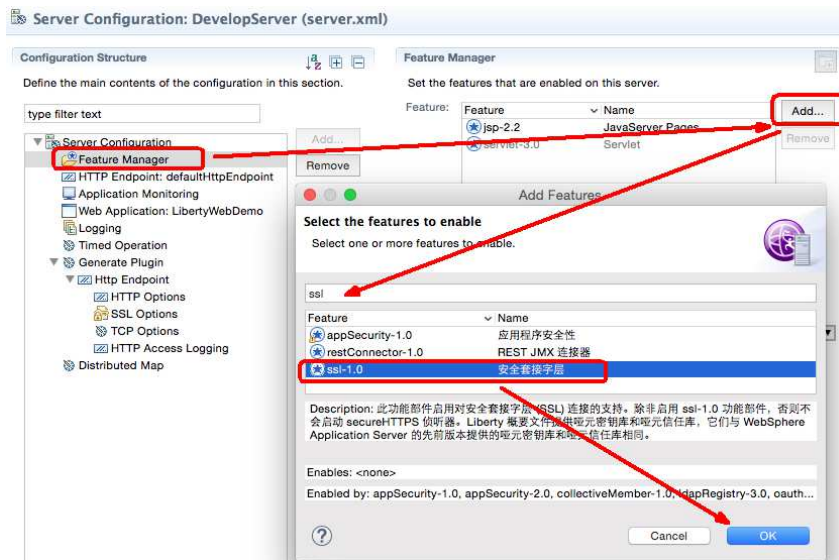
观察日志，可以看到如下信息：

00000024 com.ibm.ws.ssl.config.WSKeyStore A CWPKI0803A: 已在 7.083 秒内创建 SSL 证书。SSL 密钥文件：resources/security/key.jks

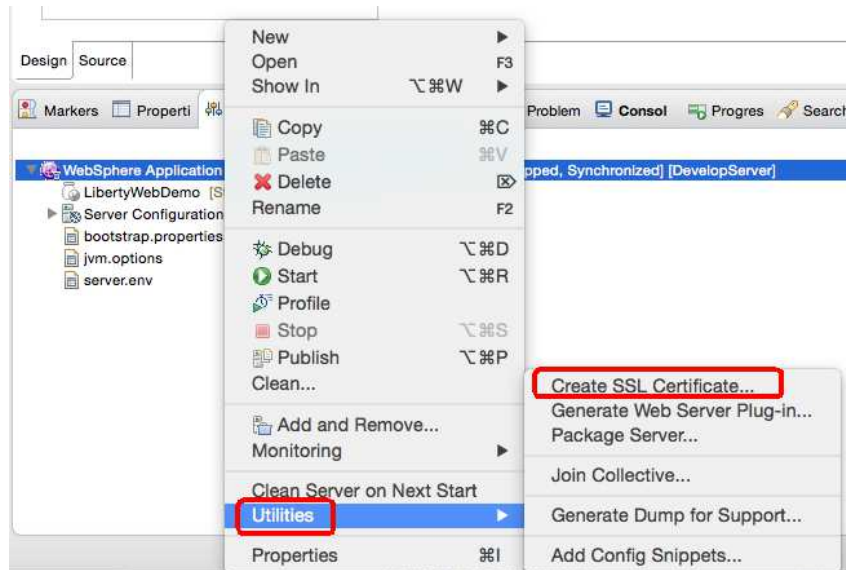
访问页面，结果如下：

9.3.4. 通过 Eclipse 开发工具配置

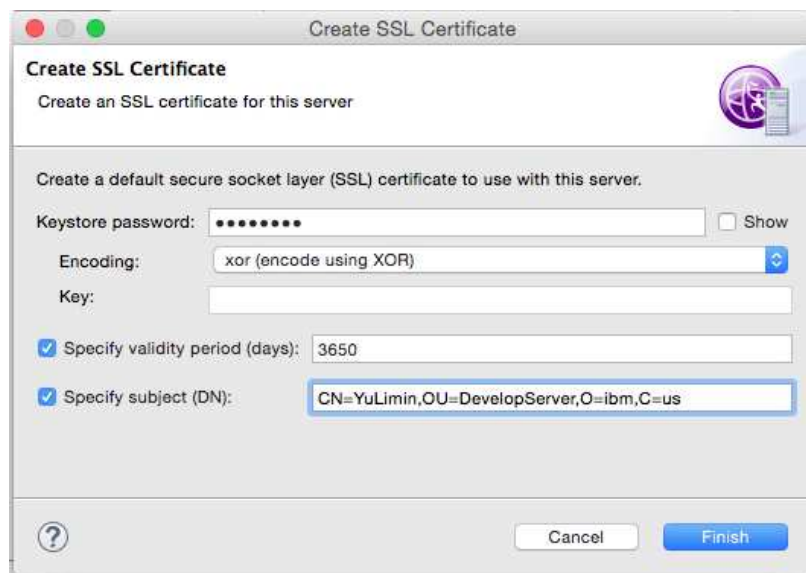
添加 SSL 功能特性，如下：



右键 Liberty 服务器，Utilities → Create SSL Certificat ...



输入 Keystore 密码，选择加密方式，指定有效时间，指定 DN，然后 Finish 即可生成，如下图所示：



最后将生成的配置信息拷贝到 `server.xml` 配置文件中即可。

9.4. 集群配置与管理

首先需要安装 collectiveController-1.0 特性组件，安装示例如下：

featureManager install collectiveController-1.0

安装过程输出信息如下：

正在与 IBM WebSphere Liberty Repository 联系...

其他功能部件的条款和条件：

单击“我同意”按钮，即表示被许可方同意将要下载的程序代码、样本、更新、修订及密钥和文件之类的相关许可材料（统称“代码”）受被

许可方在获取相关程序（下载的代码适用的程序）时所接受的许可协议的条款的限制。被许可方还同意将此代码仅作为自己具有有效协议或权

利证明的程序的一部分进行安装或使用。术语“程序”和“权利证明”在

IBM 国际程序许可协议（“IPLA”）中的含义相同。要了解 IPLA，可参阅：

<http://www.ibm.com/software/sla/>

如果被许可方获取了这些组件用于免费产品（非试用产品），那么只能在开发人员机器或构建服务器上使用这些组件。

“开发人员机器”定义为运行主要操作系统和程序的物理或虚拟桌面环境，该环境仅允许一个（1）

个指定的开发人员访问和使用。物理或虚拟桌面环境包括本地和远程云环境。在开发人员机器上，被许可方未被授权使用本程序处理生产工作

负载，模拟生产工作负载或测试任何代码、应用程序或系统的可扩展性。

选择 “[1] 我同意”，或 “[2] 我不同意”： **1**

第 1 个步骤（共 7 个步骤）：正在启动安装...

第 2 个步骤（共 7 个步骤）：正在检查功能部件...

第 3 个步骤（共 7 个步骤）：正在解析远程功能部件。此过程可能要用若干分钟完成。如果在三分钟之后，该过程仍在运行，那么请重新启动该过程。

第 4 个步骤（共 7 个步骤）：正在下载 collectiveController-1.0...

第 5 个步骤（共 7 个步骤）：正在安装 collectiveController-1.0...

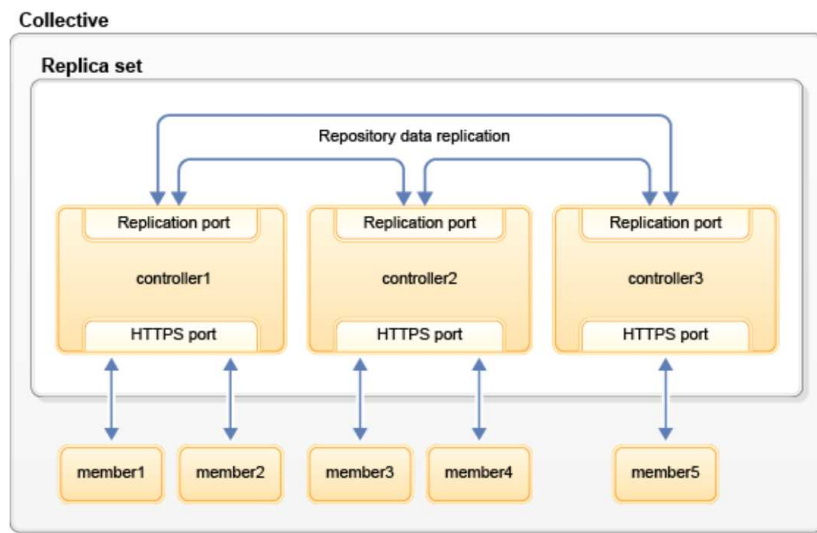
第 6 个步骤（共 7 个步骤）：正在清除临时文件...

第 7 个步骤（共 7 个步骤）：安装完成

CWWKF1017I: 已成功安装一个或多个功能部件: [collectiveController-1.0]。

产品验证已成功完成。

接下来，我们将要进行如下的集群拓扑示例图配置过程：



9.4.1. 创建集合（Collective）

集合是一个独立管理域中所有 Liberty 服务器的统称。一个集合由以下部分组成：至少一个“集合控制器”（Collective Controller），这个集合控制器就是一个启用了 collectiveController-1.0 功能特性的 Liberty 服务器；一个集合也可以包含多个“集合成员”（Collective Member），它启用了 collectiveMember-1.0 功能特性。

一个集合可以配置多个集合控制器，称之为复制集（Replica Set），构建控制器的高可用性。

9.4.1.1. 创建集合控制器服务器

创建名字为 **controller1** 的控制器：

```
server create controller1
```

服务器 **controller1** 已创建。

9.4.1.2. 创建集合控制器配置信息

它将建立管理域的安全配置信息：

```
collective create controller1 --keystorePassword=P@ssw0rd
```

说明，为了方便记忆与统一操作，本文凡涉及到设置密码的均设置为 **P@ssw0rd**

注意主机名的解析与配置信息，配置时将检测 Fully Qualified Domain Name (FQDN)，要不然会报错，比如：找不到所指定的服务器 controller1，等，创建过程信息如下：

正在创建所需证书以建立集合体...

这可能需要一些时间。

已成功生成控制器根证书。

已成功生成成员根证书。

已成功生成服务器身份证书。

已成功生成 HTTPS 证书。

已成功为 **controller1** 设置集合体控制器配置

请将以下各行添加至 server.xml 以启用：

```
<featureManager>
  <feature>collectiveController-1.0</feature>
</featureManager>

<!-- Define the host name for use by the collective.
      If the host name needs to be changed, the server should be
      removed from the collective and re-joined or re-replicated. -->
<variable name="defaultHostName" value="<YourServerHostName>" />

<!-- TODO: Set the security configuration for Administrative access -->
<quickStartSecurity userName="" userPassword="" />

<!-- clientAuthenticationSupported set to enable bidirectional trust -->
<ssl id="defaultSSLConfig"
      keyStoreRef="defaultKeyStore"
      trustStoreRef="defaultTrustStore"
      clientAuthenticationSupported="true" />

<!-- inbound (HTTPS) keystore -->
<keyStore id="defaultKeyStore" password="{xor}Dx8sLChvLTs="
      location="{server.config.dir}/resources/security/key.jks" />

<!-- inbound (HTTPS) truststore -->
<keyStore id="defaultTrustStore" password="{xor}Dx8sLChvLTs="
      location="{server.config.dir}/resources/security/trust.jks" />

<!-- server identity keystore -->
<keyStore id="serverIdentity" password="{xor}Dx8sLChvLTs="
      location="{server.config.dir}/resources/collective/serverIdentity.jks" />

<!-- collective trust keystore -->
<keyStore id="collectiveTrust" password="{xor}Dx8sLChvLTs="
      location="{server.config.dir}/resources/collective/collectiveTrust.jks" />

<!-- collective root signers keystore -->
<keyStore id="collectiveRootKeys" password="{xor}Dx8sLChvLTs="
      location="{server.config.dir}/resources/collective/rootKeys.jks" />
```

请确保已为服务器配置管理安全性。
需要有管理用户向集合体连接成员。

提示：同时增加管理控制台的组件信息到配置文件中，如下 adminCenter 配置：

```
<feature>adminCenter-1.0</feature>
```

将以上的信息增加到 server.xml 配置文件中，并将

```
<variable name="defaultHostName" value="<YourServerHostName>" />
```

<YourServerHostName>会自动识别出你所在的服务器的名称，如果名称不对，则需要先检查操作系统的 hosts 配置文件或者 DNS 服务器配置的名称。

再将

```
<quickStartSecurity userName="" userPassword="" />
```

根据需要设置为：

```
<quickStartSecurity userName="admin" userPassword="P@ssw0rd" />
```

提示：

但是，P@ssw0rd 这个密码虽然可以是明文的，但是显然是不合适的；可以进行通过执行 Liberty 下 bin 目录下的 securityUtility 工具来加密这个密码 P@ssw0rd，然后得到加密后的密码，比如：

```
securityUtility encode P@ssword
```

可以得到

```
{xor}Dx8sLCgwLTs=
```

然后设置为

```
<quickStartSecurity userName="admin" userPassword="{xor}Dx8sLCgwLTs=" />
```

9.4.1.3. 启动控制器服务器

server start controller1

正在启动服务器 **controller1**。
服务器 **controller1** 已启动。

并通过启动的日志 **messages.log** 进行确认启动成功，详细的日志如下：

```
A CWWKE0001I: 已启动服务器 controller1。
A CWWKE0100I: 此产品允许用于开发，但限制生产使用。可在此处查看完整的许可条款：
https://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/license/base_ilan/ilan/8.
5.5.5/lafiles/zh.html
I CWWKE0002I: 内核在 2.087 秒 后启动
I CWWKF0007I: 功能部件更新已启动。
I CWWKS0007I: 安全服务正在启动...
A CWWKZ0058I: 正在监视应用程序的 dropins。
I CWWKO0219I: TCP 通道 defaultHttpEndpoint 已启动并且正在侦听主机 demowinxp
(IPv4: 192.168.1.216) 端口 9080 上的请求。
I DYNA1001I: 名为 baseCache 的 WebSphere 动态高速缓存实例已成功初始化。
I DYNA1071I: 正使用高速缓存提供程序 default。
I DYNA1056I: 动态高速缓存（对象高速缓存）已成功初始化。
I CWWKO0219I: TCP 通道 defaultHttpEndpoint-ssl 已启动并且正在侦听主机 demowinxp
(IPv4: 192.168.1.216) 端口 9443 上的请求。
I CWWKS0008I: 安全服务已准备就绪。
I CWWKS4105I: LTPA 配置将在 0.142 秒之后准备就绪。
I CWWKX1015I: 对于 Admin Center，FILE 持久性层已初始化。
I SRVE0169I: 正在装入 Web 模块：IBMJMXConnectorREST。
I SRVE0250I: Web 模块 IBMJMXConnectorREST 已绑定到 default_host。
A CWWKT0016I: Web 应用程序可用 (default_host) :
http://demowinxp:9080/IBMJMXConnectorREST/
I CWWKX0103I: JMX REST 连接器正在运行并且是在以下服务 URL 处提供的：
service:jmx:rest://demowinxp:9443/IBMJMXConnectorREST
I CWWKX0103I: JMX REST 连接器正在运行并且是在以下服务 URL 处提供的：
service:jmx:rest://demowinxp:9443/IBMJMXConnectorREST
I SRVE0169I: 正在装入 Web 模块：ibm/api。
I SRVE0250I: Web 模块 ibm/api 已绑定到 default_host。
A CWWKT0016I: Web 应用程序可用 (default_host): http://demowinxp:9080/ibm/api/
I CWWKX6012I: 集合体控制器暂时不可用，可能是因为副本集中的更改。它应在几秒后重
新可用。当前的活动副本集为 []。已配置的副本集为 [192.168.1.216:10010]。
I CWWKX6013I: 集合体控制器状态为 {S_ACCEPTOR}，最后提出的命令为 -1，最后接受
的命令为 47，最后执行的命令为 0，日志为 47。
```

I SRVE0169I: 正在装入 Web 模块: The Liberty Admin Center。
I SRVE0250I: Web 模块 The Liberty Admin Center 已绑定到 default_host。
A CWWKT0016I: Web 应用程序可用 (default_host): http://demowinxp:9080/adminCenter/
A CWWKF0012I: 服务器已安装下列功能部件: [servlet-3.0, jsp-2.2, ssl-1.0, jndi-1.0, collectiveMember-1.0, collectiveController-1.0, json-1.0, adminCenter-1.0, distributedMap-1.0, jaxrs-1.1, restConnector-1.0]
I CWWKF0008I: 已在 2.704 秒内完成功能部件更新。
A CWWKF0011I: 服务器 controller1 已准备就绪, 可开始运行智慧地球。
I CWWKX6009I: 集合体控制器已成功连接至副本 192.168.1.216:10010。当前的活动副本集为 [192.168.1.216:10010]。已配置的副本集为 [192.168.1.216:10010]。已连接的备用副本为 []。
I SRVE0169I: 正在装入 Web 模块: The Liberty Deploy Tool。
I SRVE0250I: Web 模块 The Liberty Deploy Tool 已绑定到 default_host。
A CWWKT0016I: Web 应用程序可用 (default_host) : http://demowinxp:9080/ibm/adminCenter/deploy-1.0/
I SRVE0169I: 正在装入 Web 模块: The Liberty Explore Tool。
I SRVE0250I: Web 模块 The Liberty Explore Tool 已绑定到 default_host。
A CWWKT0016I: Web 应用程序可用 (default_host) : http://demowinxp:9080/ibm/adminCenter/explore-1.0/
I CWWKX6014I: 此集合体控制器副本已完成与其他副本的数据同步。日志是 48。
I CWWKX6011I: 集合体控制器已就绪, 可接受请求。引导者为 192.168.1.216:10010。当前的活动副本集为 [192.168.1.216:10010]。已配置的副本集为 [192.168.1.216:10010]。
I CWWKX9049I: RepositoryConfiguration MBean 可用。
I CWWKX9000I: CollectiveRepository MBean 可用。
I CWWKX7200I: ServerCommands MBean 可用。
I CWWKX1015I: 对于 Admin Center, COLLECTIVE 持久性层已初始化。
I CWWKX1000I: SingletonMessenger MBean 可用。
I SESN8501I: 会话管理器找不到持久性存储位置; 会将 HttpSession 对象存储在本地应用程序服务器的内存中。
I CWWKX9003I: CollectiveRegistration MBean 可用。
I CWWKX8114I: 服务器的路径已成功发布至集合体存储库。
I CWWKX8122I: 缺省 SSH 基于密钥的配置将用作主机认证配置。
I CWWKX8123I: 此服务器的远程主机认证已配置为 demowinxp:22, 向 ssh-key 认证用户名 AdministratorYuLimin。
I CWWKX9065I: ControllerConfig MBean 可用。
I CWWKX7912I: 已成功更新 FileServiceMXBean 属性 ReadList。
I CWWKX7912I: 已成功更新 FileServiceMXBean 属性 WriteList。
I CWWKX9030I: ClusterManager MBean 可用。
I CWWKX8116I: 服务器 STARTED 状态已成功发布至集合体存储库。
I CWWKX8112I: 服务器的主机信息已成功发布至集合体存储库。

关键信息是要看到

I CWWKX9003I: CollectiveRegistration MBean 可用。

以上 [message.log](#) 中的信息输出内容比如多，可以仅查看 [console.log](#) 中的内容，更加简洁一些，如下：

```
Web 应用程序可用 (default_host): http://demowinxp:9080/ibm/api/
Web 应用程序可用 (default_host): http://demowinxp:9080/IBMJMXConnectorREST/
Web 应用程序可用 (default_host): http://demowinxp:9080/adminCenter/
服务器已安装下列功能部件: [servlet-3.0, jsp-2.2, ssl-1.0, jndi-1.0, collectiveMember-1.0,
collectiveController-1.0, json-1.0, adminCenter-1.0, distributedMap-1.0, jaxrs-1.1,
restConnector-1.0]
服务器 controller1 已准备就绪，可开始运行智慧地球。
Web 应用程序可用 (default_host): http://demowinxp:9080/ibm/adminCenter/deploy-1.0/
Web 应用程序可用 (default_host): http://demowinxp:9080/ibm/adminCenter/explore-1.0/
```

控制台是 <http://demowinxp:9080/adminCenter/>

9.4.1.4. 创建成员并加入集合

先创建待加入的成员 **member1**

```
server create member1
```

服务器 **member1** 已创建。

将 **member1** 加入到集合中去：

```
collective join member1 --host=demowinxp --port=9443 --user=admin  
--password=P@ssw0rd --keystorePassword=P@ssw0rd
```

加入集合控制器的过程中输出的信息如下：

正在将集合体与目标控制器 demowinxp 连接：9443...

这可能需要一些时间。

尚未与目标服务器建立 SSL 信任。

证书链信息：

证书 [0]

主体集 DN：CN=demowinxp, OU=controller1, O=ibm, C=us

发布者 DN：OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号：1,398,668,473,746

到期：20-11-30 上午 10:29

SHA-1 摘要：FC:3A:B3:16:30:A0:24:AB:53:A3:48:0F:9E:D7:2A:87:32:B4:56:39

MD5 摘要：25:CF:F6:24:27:21:97:6A:36:1D:52:2A:F7:AD:8A:EA

证书 [1]

主体集 DN：OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

发布者 DN：OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号：1,391,995,387,051

到期：40-11-25 上午 10:29

SHA-1 摘要：AC:35:68:C7:4C:0D:E9:C9:00:B3:A0:C7:5E:9A:7E:F0:B7:B8:A3:92

MD5 摘要：7B:9B:D8:3F:11:CF:2A:D8:48:AA:1F:66:E6:82:1A:DA

是否要接受以上证书链？(y/n)**y**

已成功完成对控制器的 MBean 请求。

已成功为服务器 member1 加入集合体

请将以下各行添加至 server.xml 以启用:

```
<featureManager>
  <feature>collectiveMember-1.0</feature>
</featureManager>

<!-- Define the host name for use by the collective.
      If the host name needs to be changed, the server should be
      removed from the collective and re-joined or re-replicated. -->
<variable name="defaultHostName" value="demowinxp" />

<!-- Remote host authentication configuration -->
<hostAuthInfo rpcUser="admin_user_id"
              rpcUserPassword="admin_user_password" />

<!-- Connection to the collective controller -->
<collectiveMember controllerHost="demowinxp "
                  controllerPort="9443" />

<!-- clientAuthenticationSupported set to enable bidirectional trust -->
<ssl id="defaultSSLConfig"
      keyStoreRef="defaultKeyStore"
      trustStoreRef="defaultTrustStore"
      clientAuthenticationSupported="true" />

<!-- inbound (HTTPS) keystore -->
<keyStore id="defaultKeyStore" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/security/key.jks" />

<!-- inbound (HTTPS) truststore -->
<keyStore id="defaultTrustStore" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/security/trust.jks" />

<!-- server identity keystore -->
<keyStore id="serverIdentity" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/collective/serverIdentity.jks" />

<!-- collective truststore -->
<keyStore id="collectiveTrust" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/collective/collectiveTrust.jks" />
```


将以上信息更新到 `server.xml` 文件中，但是要注意 HTTP 和 HTTPS 端口，若是在同一台机器上，则需要进行对应的更改，比如将默认的 9080, 9443 改为 **9180, 9543**

并将

```
<hostAuthInfo rpcUser="admin_user_id"
               rpcUserPassword="admin_user_password" />
```

更改为

```
<hostAuthInfo rpcUser="WinAdmin"
               rpcUserPassword="P@ssw0rd" />
```

提示：

但是，`P@ssw0rd` 这个密码虽然可以是明文的，但是显然是不合适的；可以通过执行 Liberty 下 `bin` 目录下的 `securityUtility` 工具来加密这个密码 `P@ssw0rd`，然后得到加密后的密码，比如：

```
securityUtility encode P@ssword
```

可以得到

```
{xor}Dx8sLCgwLTs=
```

然后设置为

```
<hostAuthInfo rpcUser="WinAdmin"
               rpcUserPassword="{xor}Dx8sLCgwLTs=" />
```

然后，再通过命令来启动集合成员 **member1**

```
server start member1
```

正在启动服务器 **member1**。

服务器 **member1** 已启动。

通过启动日志校验集合成员是否启动成功，无报错信息。并确认 `console.log` 信息：

服务器已安装下列功能部件：[servlet-3.0, jsp-2.2, ssl-1.0, jndi-1.0, collectiveMember-1.0, clusterMember-1.0, json-1.0, distributedMap-1.0, jaxrs-1.1, restConnector-1.0]。

服务器 **member1** 已准备就绪，可开始运行智慧地球。

Web 应用程序可用 (default_host): <http://demowinxp:9180/IBMJMXConnectorREST/>

通过以上步骤，一个基本的集合拓扑已经成功创建了。在这个拓扑中，**member1** 是集合成员，**controller1** 是集合控制器。所有的集合成员发布他们自己的信息到集合控制器上。这些信息发布到集合控制器上之后，就可以直接在控制器上进行查询，而不需要将查询的请求再转发到每一个集合成员上。

9.4.2. 执行集合操作

由于以下部分的操作需要用 Jython 语言的支持，因此需要先下载 Jython 安装包。

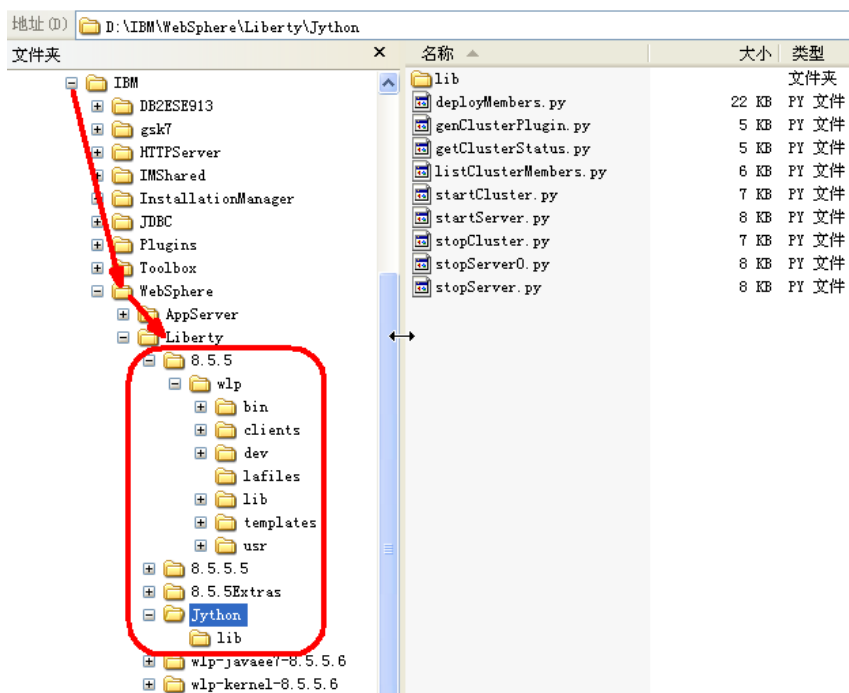
直接到 Jython 官网 <http://www.jython.org> 根据平台选择下载并安装即可。

并将附件 <http://www.Java2Class.net/IBM/WebSphere/LibertyConfig.zip> 的内容解压到 Liberty 目录下，比如：

D:\IBM\WebSphere\Liberty\Jython

D:\IBM\WebSphere\Liberty\8.5.5.7\wlp

目录结构如下：



为了获取集群状态信息以及相关的操作，设置如下系统变量，以方便进行操作：

```
set JAVA_HOME=D:\IBM\WebSphere\AppServer\8.5.5\java_1.7_32
```

```
set JAVA_HOME=D:\JDK\8.0
```

```
set PATH=%JAVA_HOME%\bin;%PATH%
```

```
set JYTHON_HOME=D:\OpenSource\Jython\2.7.0
```

```
set PATH=%JYTHON_HOME%\bin;%PATH%
```

```
set WLP_HOME=D:\IBM\WebSphere\Liberty\8.5.5.7
```

```
set CLASSPATH=%WLP_HOME%\wlp\clients\restConnector.jar
```

```
set JYTHONPATH=%WLP_HOME%\wlp\clients\jython;%WLP_HOME%\..\Jython\lib
```

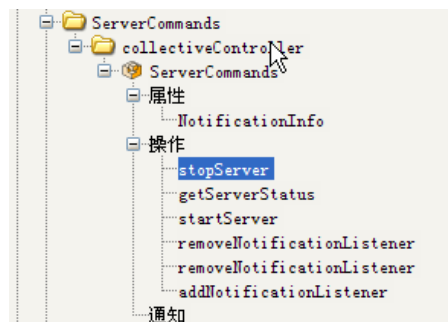
```
set ControllerHost=demowinxp
```

```
set MemberHost=demowinxp
```

通过 Jython 脚本来执行集合控制器上的 MBean 操作，从而进行停止、启动集合成员。

停止 member1 成员

调用 ServerCommands MBean 的 stopServer 操作来进行停止



```
jython %WLP_HOME%\..\Jython\stopServer.py --serverHost=%ControllerHost%
--serverUsrdir=%WLP_HOME%\wlp\usr --serverName=member1 --host=%MemberHost%
--port=9443 --user=admin --password=P@ssw0rd
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd
```

Connecting to the server...

Successfully connected to the server "demowinxp:9443"

Server stopped successfully

执行过程中可以通过 TCPView 工具查看，如果有 microsoft-ds 端口在通讯，说明是会成功的，如下图所示：

Pro...	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port	State	Se...
[System Process]	0	TCP	demowinxp	1907	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1908	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1909	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1910	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1911	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1912	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1913	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1914	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1915	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1916	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1917	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1918	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1919	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1920	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1921	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1923	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1924	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1922	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1926	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1927	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1928	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1929	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1930	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1931	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1938	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1939	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1940	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1942	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1943	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1932	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1933	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1934	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1935	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1936	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1937	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1941	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1959	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1960	demowinxp	microsoft-ds	TIME_WAIT	
[System Process]	0	TCP	demowinxp	1961	demowinxp	microsoft-ds	TIME_WAIT	
alg.exe	432	TCP	DemoWinXP	1206	DemoWinXP	0	LISTENING	
explorer.exe	860	TCP	demowinxp	1402	43.250.12.36	http	CLOSE_WAIT	
HprSnap5.exe	3916	UDP	DemoWinXP	1193	*	*		
java.exe	2968	TCP	demowinxp	1900	demowinxp	9443	ESTABLISHED	
java.exe	2968	TCP	demowinxp	1899	demowinxp	9443	ESTABLISHED	
java.exe	1678	TCP	DemoWinXP	1678	localhost	1678	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1678	localhost	1679	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1680	localhost	1681	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1681	localhost	1680	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1683	localhost	1682	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1682	localhost	1683	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1684	localhost	1685	ESTABLISHED	
javaw.exe	1728	TCP	DemoWinXP	1685	localhost	1684	ESTABLISHED	
javaw.exe	1728	TCP	demowinxp	9443	DemoWinXP	0	LISTENING	

如果没有则会报错：

如果在执行以上命令时碰到如下错误，那是在启动或停止集成员时需要远程执行和访问（Remote Execute and Access, RXA）的权限。

An exception was caught while processing the stopServer command

javax.management.MBeanException: java.net.ConnectException: CWWKX7204E: 无法使用提供的凭证连接到主机 demowinxp。

请参照如下步骤进行解决

<http://www-01.ibm.com/support/docview.wss?uid=swg21636431>

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.installation.nd.doc/ae/cins_cim_rxa_requirements.html

Commented [YuLimn1]: 注意：在 Liberty 8.5.5.5 上面应当存在 BUG，一直无法成功!!!

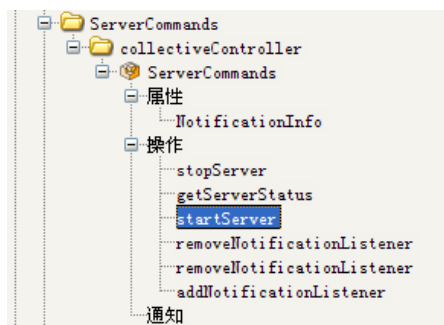
状态验证

server status member1

服务器 **member1** 未在运行。

启动 member1 成员

调用 ServerCommands MBean 的 startServer 操作来进行启动



```
jython %WLP_HOME%\..\Jython\startServer.py --serverHost=%ControllerHost%  
--serverUsrdir=%WLP_HOME%\wlp\usr --serverName=member1 --host=%MemberHost%  
--port=9443 --user=admin --password=P@ssw0rd  
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks  
--truststorePassword=P@ssw0rd
```

Connecting to the server...

Successfully connected to the server "demowinxp:9443"

Server started successfully

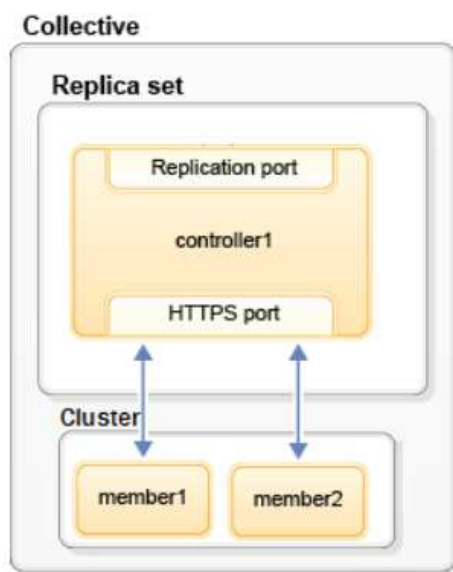
状态再次验证

server status member1

服务器 **member1** 正在运行。

9.5. 集群规划与创建

给集合增加新的成员，并将这些成员服务器加入到默认的集群当中。
并且可以通过调用集合控制器上的 MBean 操作来启动、停止集群。



如上，我们准备给默认的集群增加一个成员 **member2**

9.5.1. 创建集合成员

先创建集合成员

server create member2

服务器 **member2** 已创建。

加入到集合中

```
collective join member2 --host=demowinxp --port=9443 --user=admin  
--password=P@ssw0rd --keystorePassword=P@ssw0rd
```

```
collective join member2 --host=demowinxp --port=9443 --user=admin  
--password=P@ssw0rd --keystorePassword=P@ssw0rd --hostName=<member host>
```

正在将集合体与目标控制器 demowinxp 连接: 9443...
这可能需要一些时间。

尚未与目标服务器建立 SSL 信任。

证书链信息:

证书 [0]

主体集 DN: CN=demowinxp, OU=controller1, O=ibm, C=us

发布者 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号: 1,398,668,473,746

到期: 20-11-30 上午 10:29

SHA-1 摘要: FC:3A:B3:16:30:A0:24:AB:53:A3:48:0F:9E:D7:2A:87:32:B4:56:39

MD5 摘要: 25:CF:F6:24:27:21:97:6A:36:1D:52:2A:F7:AD:8A:EA

证书 [1]

主体集 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

发布者 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号: 1,391,995,387,051

到期: 40-11-25 上午 10:29

SHA-1 摘要: AC:35:68:C7:4C:0D:E9:C9:00:B3:A0:C7:5E:9A:7E:F0:B7:B8:A3:92

MD5 摘要: 7B:9B:D8:3F:11:CF:2A:D8:48:AA:1F:66:E6:82:1A:DA

是否要接受以上证书链? (y/n) **y**

已成功完成对控制器的 MBean 请求。

已成功为服务器 member2 加入集合体

请将以下各行添加至 server.xml 以启用:

```
<featureManager>
  <feature>collectiveMember-1.0</feature>
</featureManager>

<!-- Define the host name for use by the collective.
      If the host name needs to be changed, the server should be
      removed from the collective and re-joined or re-replicated. -->
<variable name="defaultHostName" value="demowinxp" />

<!-- Remote host authentication configuration -->
<hostAuthInfo rpcUser="admin_user_id"
              rpcUserPassword="admin_user_password" />

<!-- Connection to the collective controller -->
<collectiveMember controllerHost="demowinxp"
                  controllerPort="9443" />

<!-- clientAuthenticationSupported set to enable bidirectional trust -->
<ssl id="defaultSSLConfig"
      keyStoreRef="defaultKeyStore"
      trustStoreRef="defaultTrustStore"
      clientAuthenticationSupported="true" />

<!-- inbound (HTTPS) keystore -->
<keyStore id="defaultKeyStore" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/security/key.jks" />

<!-- inbound (HTTPS) truststore -->
<keyStore id="defaultTrustStore" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/security/trust.jks" />

<!-- server identity keystore -->
<keyStore id="serverIdentity" password="{xor}Dx8sLChvLTs="
          location="{server.config.dir}/resources/collective/serverIdentity.jks" />

<!-- collective truststore -->
<keyStore id="collectiveTrust" password="{xor}Dx8sLChvLTs="
```



```
location="${server.config.dir}/resources/collective/collectiveTrust.jks" />
```

根据加入集合时输出的信息进行更新 `server.xml`，同时注意 HTTP 和 HTTPS 端口号的修改为 9181 和 9544。

并将

```
<hostAuthInfo rpcUser="admin_user_id"
               rpcUserPassword="admin_user_password" />
```

更改为

```
<hostAuthInfo rpcUser="WinAdmin"
               rpcUserPassword="{xor}Dx8sLChvLTs=" />
```

启动集合新成员

```
server start member2
```

正在启动服务器 `member2`。

服务器 `member2` 已启动。

确认启动成功，这样我们有了两个成员服务器了。

关键信息：

`I CWWKX8055I: 集合体成员已建立与集合体控制器的连接。`

然后，可以尝试通过 Jython 脚本来停止、启动 `member2`，并确认操作成功。

提示：

第一次要先用 `server start` 之后，才可以通过 Jython 脚本进行 `start` 与 `stop` 成员，要不然就报错，比如 `Logon failure` 之类的。

9.5.2. 将集合成员分配到集群

将集合成员分配到默认的集群中去，集群配置是动态更新并发布到集合控制器中的。
增加如下配置到每个集合成员的 server.xml 配置中，上面我们创建的 **member1** 与 **member2**

```
<featureManager>
  <feature>clusterMember-1.0</feature>
</featureManager>
```

查看成员日志信息如下：

```
I CWWKF0007I: 功能部件更新已启动。
I CWWKX7400I: ClusterMember MBean 可用。
               defaultCluster
A CWWKG0017I: 服务器配置已成功地在 0.732 秒内更新。
A CWWKF0012I: 服务器已安装下列功能部件: [clusterMember-1.0]。
A CWWKF0008I: 已在 0.730 秒内完成功能部件更新。
```

集合控制器日志信息如下：

```
I CWWKX9053I: 集群 defaultCluster 已创建。
I CWWKX9051I: 服务器 member2 已添加至集群 defaultCluster。
I CWWKX9051I: 服务器 member1 已添加至集群 defaultCluster。
```

获取集群状态

```
jython %WLP_HOME%\..\Jython\getClusterStatus.py defaultCluster --host=demowinxp
--port=9443 --user=admin --password=P@ssw0rd
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd
```

返回结果如下：

```
Connecting to the server...
Successfully connected to the server "demowinxp:9443"
Status for cluster defaultCluster: STARTED
```

查看成员的状态信息

server status member1

服务器 **member1** 正在运行。

server status member2

服务器 **member2** 正在运行。

停止集群

```
jython %WLP_HOME%\..\Jython\stopCluster.py --clusterName=defaultCluster
--host=demowinxp --port=9443 --user=admin --password=P@ssw0rd
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd
```

Connecting to the server...

Successfully connected to the server "demowinxp:9443"

Successfully connected to the server "demowinxp:9443"

demowinxp,D:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member2 **stopped**, RC=0

demowinxp,D:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member1 **stopped**, RC=0

也可以通过 **server stop** 脚本来一个个手工地停止各个成员:

server stop member1

server stop member2

接下来, 通过命令来启动一下集群, 并检查成员状态信息是否正确。

启动集群

```
jython %WLP_HOME%\..\Jython\startCluster.py --clusterName=defaultCluster
--host=demowinxp --port=9443 --user=admin --password=P@ssw0rd
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd
```

Connecting to the server...

Successfully connected to the server "demowinxp:9443"

demowinxp,D:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member2 **started**, RC=0

demowinxp,D:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member1 **started**, RC=0

再次检查集群状态

```
jython %WLP_HOME%\..\Jython\getClusterStatus.py defaultCluster --host=demowinxp
--port=9443 --user=admin --password=P@ssw0rd
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd
```

Connecting to the server...

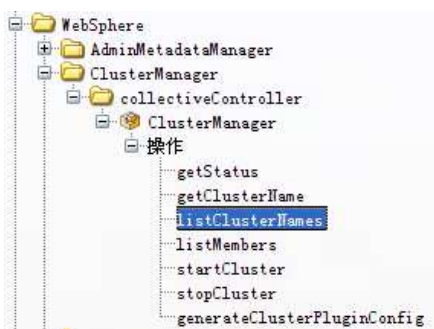
Successfully connected to the server "demowinxp:9443"

Status for cluster **defaultCluster: STARTED**

这两个服务器成员 **member1** 与 **member2** 现在是划入到了默认的 **defaultCluster** 集群中了。

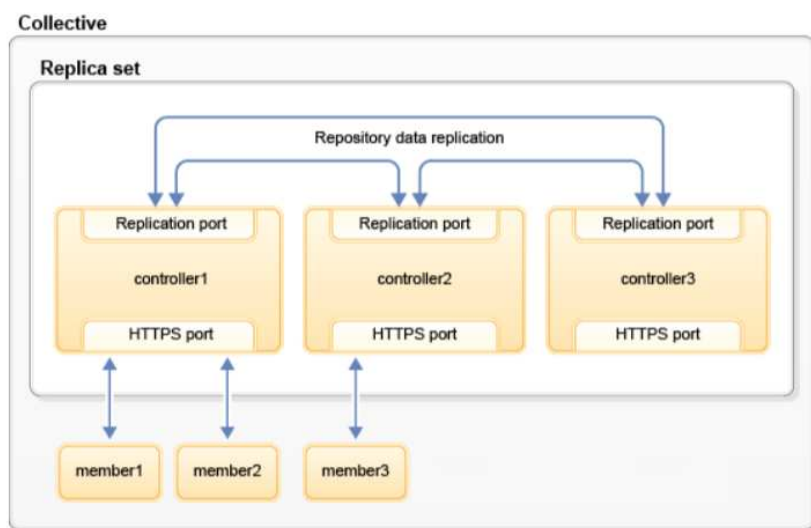
在一个集合中可以定义多个集群，但是一个服务器只能划入到某一个集群中，根据需要进行不同集群的划分即可。

ClusterManager MBean 是一个集群的操作接口，演示了集合控制器的功能，比如通过 listClusterNames、listMembers 和 getStatus 操作来进行数据信息查询。



9.6. 构建高可用的集合控制器

我们将在管理域中增加集合控制器来构建高可用的环境。每一个新的集合控制器都称之为“复本”（Replica），因为它们与源集合控制器有相同的安全配置，并且任何一个控制器的信息将自动地复制到其他所有的控制器上。一旦配置完成，在复制集中的所有的集合控制器都可以执行与源控制器相同的操作。



创建第二、第三个控制器 **controller2** 和 **controller3**

```
server create controller2
```

```
server create controller3
```

如果是在同一台机器上创建的话，则需要进行更新 `server.xml` 的 HTTP 和 HTTPS 端口号，比如： **controller2** 的修改为 9090 和 9453， **controller3** 的修改为 9091 和 9454。

复制源集合控制器的管理安全域配置信息到新的控制器 **controller2**

```
collective replicate controller2 --host=demowinxp --port=9443 --user=admin
--password=P@ssw0rd --keystorePassword=P@ssw0rd
```

在生成的 `server.xml` 中注意更新 **HTTP** 和 **HTTPS** 端口号。

正在复制目标集合体控制器 demowinxp:9443...
这可能需要一些时间。

尚未与目标服务器建立 SSL 信任。

证书链信息:

证书 [0]

主体集 DN: CN=demowinxp, OU=controller1, O=ibm, C=us

发布者 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号: 1,398,668,473,746

到期: 20-11-30 上午 10:29

SHA-1 摘要: FC:3A:B3:16:30:A0:24:AB:53:A3:48:0F:9E:D7:2A:87:32:B4:56:39

MD5 摘要: 25:CF:F6:24:27:21:97:6A:36:1D:52:2A:F7:AD:8A:EA

证书 [1]

主体集 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

发布者 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号: 1,391,995,387,051

到期: 40-11-25 上午 10:29

SHA-1 摘要: AC:35:68:C7:4C:0D:E9:C9:00:B3:A0:C7:5E:9A:7E:F0:B7:B8:A3:92

MD5 摘要: 7B:9B:D8:3F:11:CF:2A:D8:48:AA:1F:66:E6:82:1A:DA

是否要接受以上证书链? (y/n)y

已成功完成对控制器的 MBean 请求。

已成功将控制器复制为服务器 **controller2**

将以下行添加至 `server.xml` 以允许:

```
<featureManager>
  <feature>collectiveController-1.0</feature>
</featureManager>
```

```
<!-- Define the host name for use by the collective.
```

```

        If the host name needs to be changed, the server should be
        removed from the collective and re-joined or re-replicated. -->
<variable name="defaultHostName" value="demowinxp" />

<!-- Remote host authentication configuration -->
<hostAuthInfo rpcUser="admin_user_id"
    rpcUserPassword="admin_user_password" />

<!-- Configuration of the collective controller replica.
    TODO: If this replica is on the same host as the original controller,
    change the replicaPort.
    TODO: If the target controller's replica port is not null
    (the default) change the value in replicaSet. -->
<collectiveController replicaPort="10010"
    replicaSet="demowinxp:10010"
    isInitialReplicaSet="false" />

<!-- TODO: Define the security configuration exactly as defined in the
    target controller from which this was replicated. -->
<quickStartSecurity userName="" userPassword="" />

<!-- clientAuthenticationSupported set to enable bidirectional trust -->
<ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="defaultTrustStore"
    clientAuthenticationSupported="true" />

<!-- inbound (HTTPS) keystore -->
<keyStore id="defaultKeyStore" password="{xor}Dx8sLChvLTs="
    location="{server.config.dir}/resources/security/key.jks" />

<!-- inbound (HTTPS) truststore -->
<keyStore id="defaultTrustStore" password="{xor}Dx8sLChvLTs="
    location="{server.config.dir}/resources/security/trust.jks" />

<!-- server identity keystore -->
<keyStore id="serverIdentity" password="{xor}Dx8sLChvLTs="
    location="{server.config.dir}/resources/collective/serverIdentity.jks" />

<!-- collective truststore -->
<keyStore id="collectiveTrust" password="{xor}Dx8sLChvLTs="
    location="{server.config.dir}/resources/collective/collectiveTrust.jks" />

<!-- collective root signers keystore

```

TODO: set password to the collectiveRootKeys password in the original controller -->

```
<keyStore id="collectiveRootKeys" password=""  
    location="${server.config.dir}/resources/collective/rootKeys.jks" />
```

请确保已对正好作为当前集体控制器的新服务器配置管理安全性。
还应将 collectiveRootKeys 的密码设置为正确密码。

加上 HTTP 与 HTTPS 端口，一共有**五个地方**的配置信息需要修改：

```
<hostAuthInfo rpcUser="admin_user_id"  
    rpcUserPassword="admin_user_password" />
```

更改为

```
<hostAuthInfo rpcUser="WinAdmin"  
    rpcUserPassword="{xor}Dx8sLChvLTs=" />
```

```
<collectiveController replicaPort="10010"  
    replicaSet="demowinxp:10010"  
    isInitialReplicaSet="false" />
```

由于是在同一台服务器上，因此考虑到默认的 controller1 的复制端口号是 10010，因此设置 controller2 的复制端口 replicaPort 号为 10011

```
<collectiveController replicaPort="10011"  
    replicaSet="demowinxp:10010"  
    isInitialReplicaSet="false" />
```

```
<quickStartSecurity userName="" userPassword="" />
```

改为与 controller1 相同的管理用户与密码

```
<quickStartSecurity userName="admin" userPassword="{xor}Dx8sLChvLTs=" />
```

```
<keyStore id="collectiveRootKeys" password=""  
    location="${server.config.dir}/resources/collective/rootKeys.jks" />
```

改为与 controller1 相同的根证书密码

```
<keyStore id="collectiveRootKeys" password="{xor}Dx8sLChvLTs="  
    location="${server.config.dir}/resources/collective/rootKeys.jks" />
```


同样，复制源集合控制器的管理安全域配置信息到新的控制器 **controller3**

```
collective replicate controller3 --host=demowinxp --port=9443 --user=admin  
--password=P@ssw0rd --keystorePassword=P@ssw0rd
```

正在复制目标集合体控制器 demowinxp:9443...
这可能需要一些时间。

尚未与目标服务器建立 SSL 信任。

证书链信息:

证书 [0]

主体集 DN: CN=demowinxp, OU=controller1, O=ibm, C=us

发布者 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号: 1,398,668,473,746

到期: 20-11-30 上午 10:29

HA-1 摘要: FC:3A:B3:16:30:A0:24:AB:53:A3:48:0F:9E:D7:2A:87:32:B4:56:39

D5 摘要: 25:CF:F6:24:27:21:97:6A:36:1D:52:2A:F7:AD:8A:EA

证书 [1]

主体集 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

发布者 DN : OU=controllerRoot, O=0775a385-9a75-411b-adcf-6cc45075faf7,
DC=com.ibm.ws.collective

序列号: 1,391,995,387,051

到期: 40-11-25 上午 10:29

HA-1 摘要: AC:35:68:C7:4C:0D:E9:C9:00:B3:A0:C7:5E:9A:7E:F0:B7:B8:A3:92

D5 摘要: 7B:9B:D8:3F:11:CF:2A:D8:48:AA:1F:66:E6:82:1A:DA

是否要接受以上证书链? (y/n)y

已成功完成对控制器的 MBean 请求。

已成功将控制器复制为服务器 **controller3**

将以下行添加至 server.xml 以允许:

```
<featureManager>  
  <feature>collectiveController-1.0</feature>  
</featureManager>
```

```
<!-- Define the host name for use by the collective.  
     If the host name needs to be changed, the server should be
```

```

        removed from the collective and re-joined or re-replicated. -->
<variable name="defaultHostName" value="demowinxp" />

<!-- Remote host authentication configuration -->
<hostAuthInfo rpcUser="admin_user_id"
    rpcUserPassword="admin_user_password" />

<!-- Configuration of the collective controller replica.
    TODO: If this replica is on the same host as the original controller,
    change the replicaPort.
    TODO: If the target controller's replica port is not null
    (the default) change the value in replicaSet. -->
<collectiveController replicaPort="10010"
    replicaSet="demowinxp:10010"
    isInitialReplicaSet="false" />

<!-- TODO: Define the security configuration exactly as defined in the
    target controller from which this was replicated. -->
<quickStartSecurity userName="" userPassword="" />

<!-- clientAuthenticationSupported set to enable bidirectional trust -->
<ssl id="defaultSSLConfig"
    keyStoreRef="defaultKeyStore"
    trustStoreRef="defaultTrustStore"
    clientAuthenticationSupported="true" />

<!-- inbound (HTTPS) keystore -->
<keyStore id="defaultKeyStore" password="{xor}PDAXKy0wMzM6LWw="
    location="{server.config.dir}/resources/security/key.jks" />

<!-- inbound (HTTPS) truststore -->
<keyStore id="defaultTrustStore" password="{xor}PDAXKy0wMzM6LWw="
    location="{server.config.dir}/resources/security/trust.jks" />

<!-- server identity keystore -->
<keyStore id="serverIdentity" password="{xor}PDAXKy0wMzM6LWw="
    location="{server.config.dir}/resources/collective/serverIdentity.jks" />

<!-- collective truststore -->
<keyStore id="collectiveTrust" password="{xor}PDAXKy0wMzM6LWw="
    location="{server.config.dir}/resources/collective/collectiveTrust.jks" />

<!-- collective root signers keystore
    TODO: set password to the collectiveRootKeys password in the

```

```

original controller -->
<keyStore id="collectiveRootKeys" password=""
    location="${server.config.dir}/resources/collective/rootKeys.jks" />

```

请确保已对正好作为当前集合体控制器的新服务器配置管理安全性。
还应将 collectiveRootKeys 的密码设置为正确密码。

加上 HTTP 与 HTTPS 端口，一共有**五个地方**的配置信息需要修改：

```

<hostAuthInfo rpcUser="admin_user_id"
    rpcUserPassword="admin_user_password" />

```

更改为

```

<hostAuthInfo rpcUser="WinAdmin"
    rpcUserPassword="{xor}Dx8sLChvLTs=" />

```

```

<collectiveController replicaPort="10010"
    replicaSet="demowinxp:10010"
    isInitialReplicaSet="false" />

```

由于是在同一台服务器上，因此考虑到默认的 controller1 的复制端口号是 10010，以及 controller2 分配了 10011 端口了，因此设置 controller3 的复制端口 replicaPort 号为 10012

```

<collectiveController replicaPort="10012"
    replicaSet="demowinxp:10010"
    isInitialReplicaSet="false" />

```

```

<quickStartSecurity userName="" userPassword="" />

```

改为与 controller1 相同的管理用户与密码

```

<quickStartSecurity userName="admin" userPassword="{xor}Dx8sLChvLTs=" />

```

```

<keyStore id="collectiveRootKeys" password=""
    location="${server.config.dir}/resources/collective/rootKeys.jks" />

```

改为与 controller1 相同的根证书密码

```

<keyStore id="collectiveRootKeys" password="{xor}Dx8sLChvLTs="
    location="${server.config.dir}/resources/collective/rootKeys.jks" />

```

启动新创建的集合控制器

```
server start controller2
```

```
server start controller3
```

并验证启动日志，看是否启动成功。

注意查看是否有与其他控制器进行通讯的日志信息，如下：

controller1 日志信息如下：

```
I CWWKX6009I: 集合体控制器已成功连接至副本 192.168.1.216:10012。当前的活动副本集为 [192.168.1.216:10010]。已配置的副本集为 [192.168.1.216:10010]。已连接的备用副本为 [192.168.1.216:10011, 192.168.1.216:10012]。
```

controller2 日志信息如下：

```
I CWWKX6009I: 集合体控制器已成功连接至副本 192.168.1.216:10011。当前的活动副本集为 []。已配置的副本集为 [192.168.1.216:10010]。已连接的备用副本为
```

```
I CWWKX6009I: 集合体控制器已成功连接至副本 192.168.1.216:10010。当前的活动副本集为 [192.168.1.216:10010]。已配置的副本集为 [192.168.1.216:10010]。已连接的备用副本为 [192.168.1.216:10011]。
```

controller3 日志信息如下：

```
I CWWKX6009I: 集合体控制器已成功连接至副本 192.168.1.216:10012。当前的活动副本集为 []。已配置的副本集为 [192.168.1.216:10010]。已连接的备用副本为 [192.168.1.216:10012]。
```

```
I CWWKX6009I: 集合体控制器已成功连接至副本 192.168.1.216:10010。当前的活动副本集为 [192.168.1.216:10010]。已配置的副本集为 [192.168.1.216:10010]。已连接的备用副本为 [192.168.1.216:10012]。
```

更新控制器 **controller2** 配置信息

```
jython %WLP_HOME%\..\Jython\updateRepositoryConfig.py add --host=demowinxp
--port=9443 --user=admin --password=P@ssw0rd
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd --endpoint=demowinxp:10011
Connecting to the server...
Successfully connected to the server "demowinxp:9443"
Adding replica endpoint yuliminmbp:10011 to the repository configuration
Operation: True
```

更新控制器 **controller3** 配置信息

```
jython %WLP_HOME%\..\Jython\updateRepositoryConfig.py add --host=demowinxp
--port=9443 --user=admin --password=P@ssw0rd
```

```
--truststore=%WLP_HOME%\wlp\usr\servers\controller1\resources\security\trust.jks
--truststorePassword=P@ssw0rd --endpoint=demowinxp:10012
Connecting to the server...
Successfully connected to the server "demowinxp:9443"
Adding replica endpoint demowinxp:10012 to the repository configuration
Operation: True
```

通过新的控制器 **controller2** 来增加一个新的集合成员 **member3**，创建 Liberty 服务器

```
server create member3
```

服务器 **member3** 已创建。

```
collective join member3 --host=demowinxp --port=9453 --user=admin
--password=P@ssw0rd --keystorePassword=P@ssw0rd
```

修改 HTTP 与 HTTPS 端口，并添加到默认集群当中

```
<feature>clusterMember-1.0</feature>
```

然后，启动新的集合成员 **member3**

```
server start member3
```

这样，我们一个默认的集群当中有了三个集群成员，接下来就可以进行应用的部署与前端的负责均衡分发了。

9.7. 部署应用到集群

更新各个集合成员服务器的 `server.xml` 配置文件信息，增加

```
<remoteFileAccess>
  <writeDir>${server.config.dir}</writeDir>
</remoteFileAccess>
```

为了部署应用分发应用程序之用，要不然无法写入文件，会发生错误：

Pushing the application to server

demowinxp,D:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr/member2

十二月 11, 2015 12:28:54 上午

com.ibm.ws.jmx.connector.client.rest.internal.FileTransferClient uploadFile(String
localSourceFile, String remoteTargetFile, boole
an expandOnCompletion)

严重: CWWKX7204E: 无法使用提供的凭证连接到主机 demowinxp。

An exception was caught while processing member

demowinxp,D:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr/member2

java.net.ConnectException: CWWKX7204E: 无法使用提供的凭证连接到主机
demowinxp。

严重: CWWKX0262E: The file operation 写 on file system path

/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr/servers/member1/apps was denied because the file
system path was not listed in the file operation white list.

An exception was caught while processing member

yuliminmbp,/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr/member1

java.io.IOException: CWWKX0262E: The file operation 写 on file system path
/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr/servers/member1/apps was denied because the file
system path was not listed in the file operation white list.

部署应用到集群中

接下来，通过控制器 **controller3** 来部署应用

```
jython $WLP_HOME/./Jython/deployAppToCluster.py
$WLP_HOME/./Jython/DefaultWebApplication.war --clusterName=defaultCluster
--host=yuliminmbp --port=9454 --user=admin --password=P@ssw0rd
--truststore=$WLP_HOME/wlp/usr/servers/controller3/resources/security/trust.jks
--truststorePassword=P@ssw0rd
```

Connecting to the server...

Successfully connected to the server "yuliminmbp:9454"

Pushing the application to server
yuliminmbp:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member3

Pushing the application to server
yuliminmbp:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member2

Pushing the application to server
yuliminmbp:/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr,member1

提示部署操作成功！

实际上是将应用程序先拷贝到各个集群成员的 `apps` 目录下面。然后仍需要在各个集群成员的配置文件中增加如下应用部署的声明：

```
<application type="war" id="DefaultWebApplication" name="DefaultWebApplication"
location="${server.config.dir}/apps/DefaultWebApplication.war"/>
```

集群各个成员在自动加载配置后，就可以根据各个不同 Liberty 集群成员的服务器监听端口进行访问了，如下所有的三个成员访问链接。

<http://yuliminmbp:9180/DefaultWebApplication/snoop>

<http://yuliminmbp:9181/DefaultWebApplication/snoop>

<http://yuliminmbp:9182/DefaultWebApplication/snoop>

通用配置技巧

为每一个集群成员创建一个 `bootstrap.properties` 配置文件，这个配置文件将包含每个集群成员独立的配置信息，比如端口号、主机名。这样就可以所有集群成员共用一个 `server.xml` 文件而减少每个配置文件进行编辑的过程。`bootstrap.properties` 配置示例如下：

```
host.name=localhost
http.port=9081
https.port=9444
keystore.password=member1
```

然后同理给其他的成员创建类似的文件并进行内容修改。

这样就可将集群中所有成员的 `server.xml` 文件进行了统一，然后不同的内容由 `bootstrap.properties` 配置文件来指定，并在启动时加载即可。

```
<httpEndpoint id="defaultHttpEndpoint" host="${httpEndpoint.host}"
  httpPort="${httpEndpoint.httpPort}" httpsPort="${httpEndpoint.httpsPort}" />
<variable name="defaultHostName" value="${variable.defaultHostName}" />
<keyStore id="defaultKeyStore" password="${keyStore.password}"
```

```
        location="${server.config.dir}/resources/security/key.jks" />
<keyStore id="defaultTrustStore" password="${keyStore.password}"
    location="${server.config.dir}/resources/security/trust.jks" />
<keyStore id="serverIdentity" password="${keyStore.password}"
    location="${server.config.dir}/resources/collective/serverIdentity.jks" />
<keyStore id="collectiveTrust" password="${keyStore.password}"
    location="${server.config.dir}/resources/collective/collectiveTrust.jks" />
```


9.8. 生成插件

在各个集群成员配置成功之后，同时确认了可以通过各台服务器的端口进行直接访问各个已部署的应用程序后；接下来就可以在这些集群服务器之前增加一个集中的入口点了，比如 IHS 等 Web 服务器。当然，第一步是要生成插件，然后提供给 IHS 等进行加载然后请求分发。

通过脚本生成插件

```
jython $WLP_HOME/../Jython/genClusterPlugin.py defaultCluster --host=yuliminmbp
--port=9454 --user=admin --password=P@ssw0rd
--truststore=$WLP_HOME/wlp/usr/servers/controller3/resources/security/trust.jks
--truststorePassword=P@ssw0rd
```

Connecting to the server...

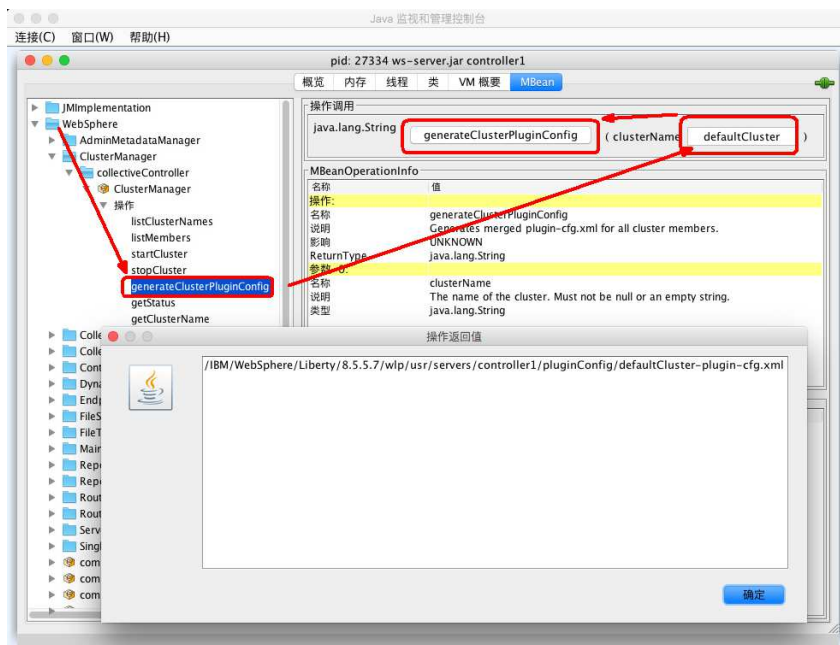
Successfully connected to the server "yuliminmbp:9454"

Generated plugin-cfg.xml file:

/IBM/WebSphere/Liberty/8.5.5.7/wlp/usr/servers/controller3/pluginConfig/defaultCluster-plugin-cfg.xml

This file will reside in the controller's host filesystem

也可以通过 JConsole 来生成插件，需要输入默认的 defaultCluster 作为参数，如下图所示：



然后通过修改 IHS 配置 `httpd.conf` 配置文件，修改 `WebSpherePluginConfig` 来加载所生成的 `defaultCluster-plugin-cfg.xml`，然后启动 IHS 进行分发：

For Liberty Cluster

LoadModule was_ap22_module D:\IBM\Plugins\8.5.5\bin\mod_was_ap22_http.dll

WebSpherePluginConfig D:\IBM\WebSphere\Liberty\8.5.5.7\defaultCluster-plugin-cfg.xml

9.9. 分布式缓存

在生产环境中，需要配置 Session 复制等集群功能，来实现高可用性。因此，在这个章节中，将要进行分布式缓存的配置。

9.10. 生产环境调优

9.10.1. 禁用应用程序监控

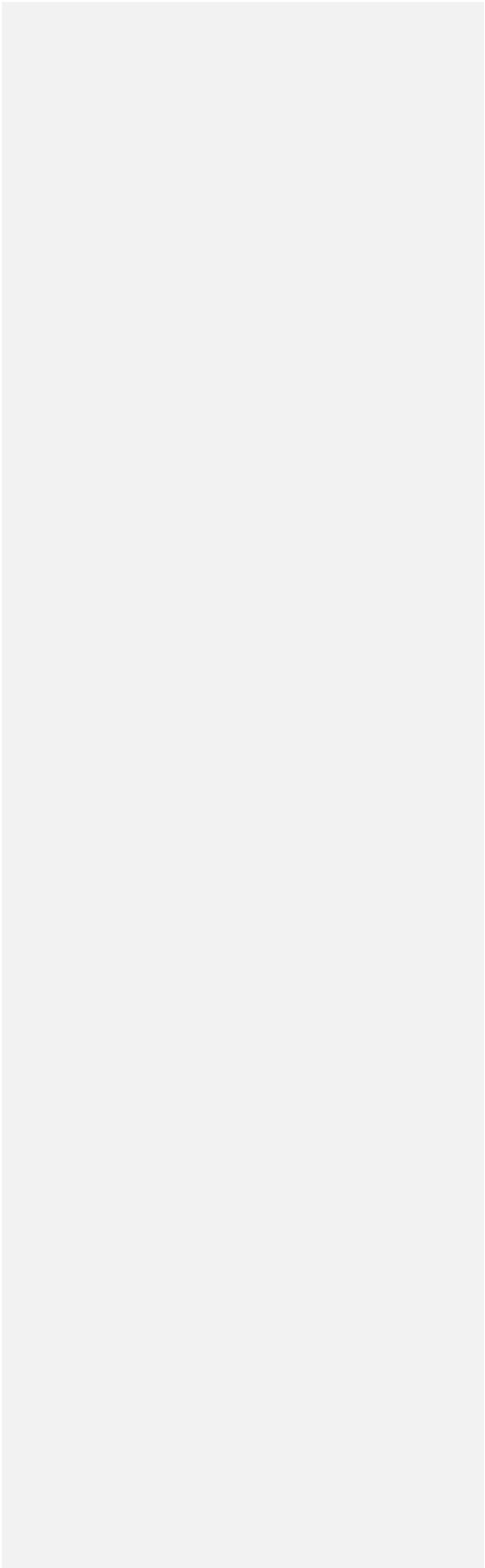
```
<applicationMonitor updateTrigger="disabled" dropinsEnabled="false"/>
```

9.10.2. 禁用配置文件监控

```
<config updateTrigger="disabled"/>
```

9.11. 生产环境最佳实践

命名规范、拓扑结构



9.12. 问题诊断

9.12.1. 日志和跟踪

默认 logs 目录下有 console.log 以及 messages.log 日志文件输出。

但是，在默认下 Liberty 没有直接启用日志跟踪功能。

messages.log 日志文件记录了服务器启动的过程。首先，启动 Liberty 内核，接着功能组件管理器开始进行初始化，并开始读取 server.xml 中的配置；这样 Liberty 服务器将在指定的端口（默认 9080、9043）开始进行监听；最后开始启动已经部署的应用程序，并对外提供服务。

通过在 server.xml 文件中增加 logging 配置即可进行详细的配置。

```
<logging          traceFormat="ADVANCED"          maxFileSize="23"          maxFiles="13"
consoleLogLevel="WARNING"></logging>
```

详细的配置说明，可以参见《Liberty profile: Trace and logging》

http://www-01.ibm.com/support/knowledgecenter/SSRTLW_9.0.0/com.ibm.websphere.wlp.nd.mutlplatform.doc/ae/rwlp_logging.html

通过 Eclipse 开发工具也可以进行图形化与源文件编辑进行配置，如下图所示：

通过 server.xml 配置文件当中的 logging 来启用日志跟踪是简单并快速的方法。但是，我们仍需要通过 bootstrap.properties 进行高级日志跟踪，以满足其他场景下的日志跟踪。比如：当碰到 Liberty 自身启动的问题时，我们需要在 Liberty 加载配置文件之前进行日志跟踪；当多个 Liberty 服务器实例共享一个 server.xml 配置文件时，而日志跟踪仅需要在某一台 Liberty 服务器实例上时。

以下为 bootstrap.properties 示例配置文件示例，它将记录所有 **net.java2class.*** 的日志信息。

```
# New trace messages are written to the end of the file (true or false)
com.ibm.ws.logging.trace.append = "true"
# maximum size of each trace file (in MB)
com.ibm.ws.logging.trace.max.file.size = 2
# maximum number of trace files
com.ibm.ws.logging.trace.max.files = 2
# Trace format (basic or advanced)
com.ibm.ws.logging.trace.format = "basic"
# Trace settings string - this string enables all trace
com.ibm.ws.logging.trace.specification = net.java2class.*=debug
```

9.12.2. 服务器内存 Dump

我们可以使用 `dump` 命令来捕捉 Liberty 服务器的状态信息，有助于 Liberty 服务器的问题诊断与分析。

直接执行 **`server dump`** 即可

通过 `dump` 命令生成的结果文件默认会存在 Liberty 服务器配置文件同级的目录下，这个文件中会包含服务器的配置文件、日志文件以及所部署的应用程序的详细信息等。无论 Liberty 服务处于运行状态还是停止状态，均可以通过 `dump` 命令来搜集这些资料，但是如果处于运行状态时，则可以搜集更多的信息：

服务器中各个 OSGi bundle 的状态；

服务器中各个 OSGi bundle 装配信息；

服务组件运行时（Service Component Runtime，SCR）所管理的组件列表；

来自 SCR 的每一个组件的详细信息；

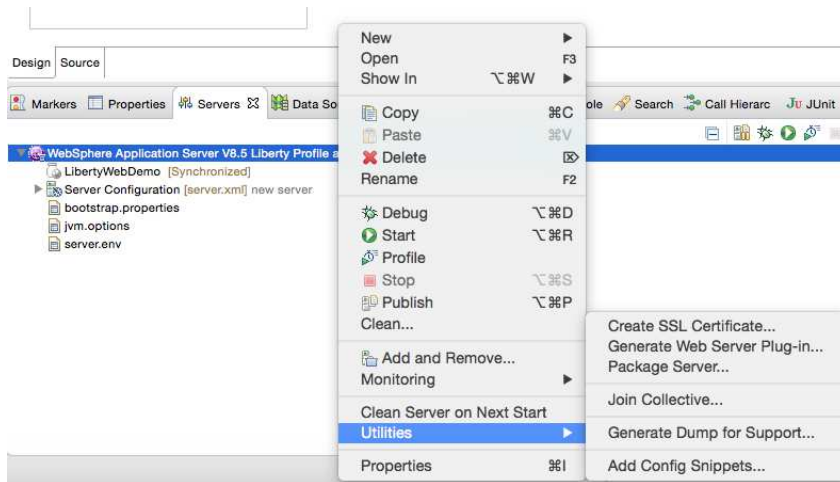
系统中仍在运行的所有请示的当前状态（需要启用 `requestTiming-1.0` 功能）；

服务器中每个 JDBC 请求的统计信息（需要启用 `timedOperations-1.0` 功能）；

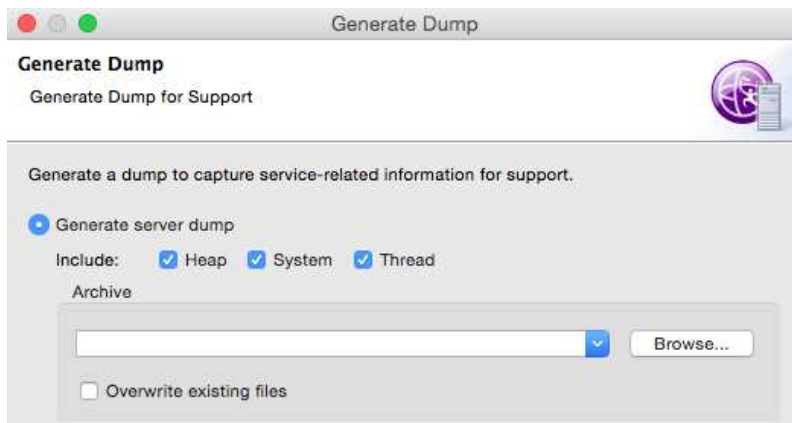
可以通过 **`-archive`** 选项则可以将生成的结果文件存放到指定的目录文件中去，如下：

`server dump -archive=/tmp/dump.zip`

在 Eclipse 开发工具中，可以直接通过右键菜单来生成，在 Liberty 服务器上右键后，在弹出来的菜单中选择“Utilities” → “Generate Dump for Support...”，即可，如下图所示：



而且可以选择是否生成“Heap”、“System”、“Thread”，如下图所示：



9.12.3.MBeans 及 JConsole

Liberty 服务器在运行时通过 JMX 将相关的信息暴露给管理员、开发人员进行管理与监控，通过 JMX 调用 MBeans 或者 JConsole 工具直接进行管理与监控。

Liberty 也提供 monitor-1.0 特性，允许用户跟踪 Liberty 运行时的信息，比如 JVM、Web 应用程序、线程池等。只要在 server.xml 配置文件中启用 monitor-1.0 就启用这个监控功能。MBeans 所提供的内容如下：

```
WebSphere:type=JvmStats
WebSphere:type=ServletStats,name=*
WebSphere:type=ThreadPoolStats,name=Default Executor
[8.5.5.0 or later]org.apache.cxf:type=WebServiceStats,service=*,port=*
[8.5.5.0 or later]WebSphere:type=SessionStats,name=*
[8.5.5.0 or later]WebSphere:type=ConnectionPool,name=*
```

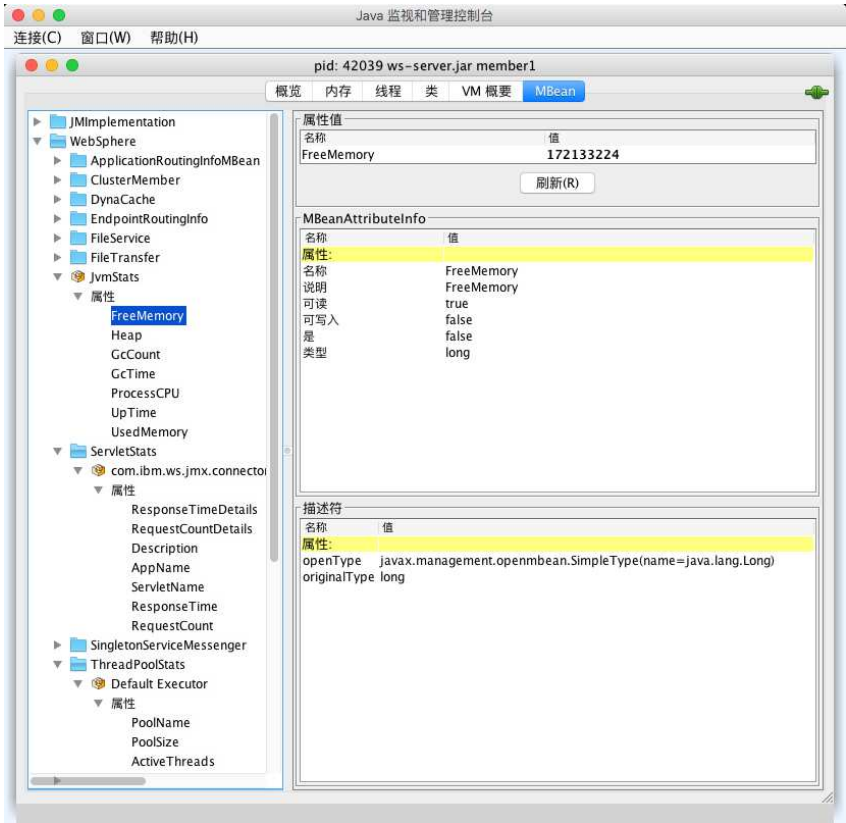
更多关于在 Liberty 中使用 MBeans，请参考：

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_mon.html

JConsole 作为 JDK/JRE 的内置工具，可以很方便地进行使用，在 JAVA_HOME\bin 目录下就可以直接启动 JConsole 工具了，如下图所示：



JConsole 启动后，若是在本机启动，则直接选择所要监控的相关进程即可，如下图所示：



上图展示了启用 monitor-1.0 特性后相关的 MBeans 内容。

9.12.4. OSGi 调试控制台

9.12.4.1. 启用 OSGi 控制台

Liberty 服务器通过 Eclipse Equinox 实现了 OSGi 核心规范，通过 `osgiConsole-1.0` 功能来帮助我们管理 OSGi 应用程序和特性，同时也可以用来调试程序。可以利用它来显示 `bundle`、`package` 和 `service` 的信息，有助于开发产品的扩展功能。

配置 OSGi 控制台，先在 `server.xml` 配置文件的 `featureManager` 段增加 `osgiConsole-1.0` 功能特性：

```
<feature>osgiConsole-1.0</feature>
```

然后在 Liberty 服务器的 `bootstrap.properties` 配置文件中设置 OSGi 控制台的端口：

```
osgi.console=8888
```

配置好 OSGi 功能并重新启动 Liberty 服务器后，可以通过 `telnet` 访问 OSGi 控制台：

```
C:\>telnet localhost 8888
Connecting To localhost
osgi>
```

输入 `help` 命令可以查看 OSGi 控制台的命令操作列表。比如，输入：

`ss` 将显示所有已安装的 `bundle` 的状态信息；

```
osgi> ss
"Framework is launched."
id      State      Bundle
0        ACTIVE      org.eclipse.osgi_3.10.2.cl50520150305-2202
                        Fragments=8
1        ACTIVE      com.ibm.ws.logging.osgi_1.0.8.cl50520150305-2202
```

`gc` 将执行一次垃圾回收的操作

```
osgi> gc
Total memory:33595392
Free memory before GC:16212104
Free memory after GC:20462992
Memory gained with GC:4250888
```

9.12.4.2. 配置 OSGi Bundle 仓库

通过以本地或远程方式共享公用的 OSGi bundle，这样 OSGi 应用程序可以访问它们。在 `server.xml` 配置文件中增加 `bundleRepository` 标签，如下配置一个本地的 OSGi 仓库：

```
<bundleRepository>
<fileset dir="directory_path" include="*.jar"/>
</bundleRepository>
```

如下配置一个远程的 OSGi 仓库：

```
<bundleRepository location="URL" />
```

URL 用来定义一个远程的 OSGi bundle 仓库，支持 HTTP、HTTPS 和 file 协议。

提示：为了在 Liberty 服务器上部署 OSGi 应用程序，需要在 `server.xml` 中启用 `blueprint-1.0`、`osgi.jpa-1.0`、`wab-1.0` 功能特性才可以。

9.12.5. 事件日志

通过新的事情日志功能特性，可以帮助我们进行调试 Liberty 应用程序，为 JDBC、Servlet、HTTP 请求之类的事件创建日志条目进行跟踪调试。这个功能特性启用日志事件和日志持续的时间。我们可以方便地控制即将被日志记录的事件类型，触发日志记录的最小间隔时间，日志的模式，以及日志样本等级。这个特性给我们提供了诊断应用程序中运行比较慢的组件的切入点，同时也调优应用程序以及它们的运行时环境的一个切入点。

事件日志功能并没有直接包含在默认的 Liberty 运行时环境当中，需要通过 featureManager 命令来安装

`featureManager install eventLogging-1.0`

安装之后，在 server.xml 配置文件的 featureManager 段增加 eventLogging-1.0 功能特性：

```
<feature>eventLogging-1.0</feature>
```

配置示例如下：

```
<eventLogging eventType="all" logMode="entryExit" sampleRate="2" minDuration="1000"
includeContextInfo="false" />
```

eventType：可以根据这个属性来指定哪些类型的事件将放记录。可以记录一个或者多个的事情类型，各个类型通过逗号（,）进行隔开即可。默认是所有的类型都将被记录。

logMode：可以根据这个属性来指定事件记录的类型：进入事件 **entry**、退出事件 **exit** 或才两者都记录。

因此，这个属性可以设置的值有 **entry**、**exit** 或者 **entryExit** 三个可选值。

sampleRate：指定每 5 个请求的事件从第几个开始记录，默认值为 1，表示所有的请求都将被记录。可用值为 1, 2, 3, 4, 5

minDuration：可以根据这个属性设置记录事件最小的间隔时间。默认值为 500 毫秒，表示，退出 **exit** 事件超过 500 毫秒就将被记录。为了记录所有的事件，则需要将之设置为 0，这个值必须是大于 0 的整数，后面可以跟上时间单位，小时 **h**，分钟 **m**，秒 **s** 以及毫秒 **ms**。没有指定时间单位，默认则为毫秒。因此，如果想指定为 1 秒，则可以设置 1000、1s 或 1000ms。

includeContextInfo：可以根据这个属性来决定是否在日志中包含有上下文信息。默认值为 **true**，可以通过设置为 **false** 将它关掉。

9.12.6. 请求访问慢以及线程挂起监测

当 Servlet 运行超过配置里所指定的时间，requestTiming-1.0 功能将提供 Servlet 请求的详细运行信息。当 Servlet 请求被检测到挂起时，它将自动收集 Java 线程堆栈，用来帮助问题诊断与分析。

启用这个功能在 server.xml 配置文件的 featureManager 段增加 requestTiming-1.0 功能特性：

```
<feature>requestTiming-1.0</feature>
```

配置示例如下：

```
<requestTiming slowRequestThreshold="300ms" hungRequestThreshold="1m" sampleRate="10"
includeContextInfo="false" />
```

slowRequestThreshold：可以根据这个属性来指定多长的时间才被认定为运行是比较慢了。可以指定值为大于 0 的整数，后面可以跟上时间单位，小时 h，分钟 m，秒 s 以及毫秒 ms。默认值是 **10 秒**。如果要禁用这个检测功能，则可以设置值为 0 即可。这个属性支持组合的时间格式，比如 1m1s 则代表运行时间超过 61 秒时就开始记录信息。

hungRequestThreshold：可以根据这个属性来指定多长的时间才被认定为线程挂起了。可以指定值为大于 0 的整数，后面可以跟上时间单位，小时 h，分钟 m，秒 s 以及毫秒 ms。默认值是 **10 分钟**。如果要禁用这个检测功能，则可以设置值为 0 即可。

sampleRate：指定请求采样的频率，默认值为 1，表示采样所有的请求。如果为了在 10 个请求中采样 1 个，则将值设置成 10 即可。

includeContextInfo：可以根据这个属性来决定是否在日志中包含有上下文信息。默认值为 true，可以通过设置为 false 将它关掉。

9.12.7.时间操作

可以通过设置预期的时间来，启用这个功能在 `server.xml` 配置文件的 `featureManager` 段增加 `requestTiming-1.0` 功能特性：

```
<feature>timedOperations-1.0</feature>
```

配置示例如下：

```
<requestTiming enableReport="ture" | false" maxNumberTimedOperations="10"
reportFrequency="1h" />
```

enableReport：这个布尔值决定是否生成记录详细信息的报告。默认值为 `true`，将详细地记录 10 个最长时间的操作，按类型进行分组，每个分组按预期的间隔时间进行排序。

maxNumberTimedOperations：这个属性用来表示当操作达到指定的次数后，将记录一个警告信息，默认值为 1000。

reportFrequency：这个属性用来指定检测的频率，按小时来生成报告，记录 10 个时间最长的操作，按类型进行分组，每个分组按平均的实际间隔时间进行排序。必须指定一个正整数，并跟随着时间单位（小时），默认值是 24 小时。

9.12.8. 如何定义整个服务器的 404、50X 之类的页面？

暂时只能在根 / 应用程序的 web.xml 中定义好之后，再到其他所有的应用里面也进行同样的定义，要不然没有办法能够完成这样的配置。

```
<error-page>
  <!-- Missing login -->
  <error-code>401</error-code>
  <location>error.jsp</location>
</error-page>
<error-page>
  <!-- Forbidden directory listing -->
  <error-code>403</error-code>
  <location>error.jsp</location>
</error-page>
<error-page>
  <!-- Missing resource -->
  <error-code>404</error-code>
  <location>error.jsp</location>
</error-page>
<error-page>
  <!-- Uncaught exception -->
  <error-code>500</error-code>
  <location>error.jsp</location>
</error-page>
<error-page>
  <!-- Unsupported servlet method -->
  <error-code>503</error-code>
  <location>error.jsp</location>
</error-page>
```

10. Docker 支持

10.1. Java EE7 & Docker 支持

http://www-01.ibm.com/common/ssi/ShowDoc.wss?docURL=/common/ssi/rep_ca/4/897/ENUS215-084/index.html

10.2. Docker 安装

首先，需要安装 Docker 环境，以下介绍 Docker 在不同平台上面的安装

10.2.1. Windows 平台下 Docker 安装

待补充。

10.2.2. Linux 平台下 Docker 安装

待补充。

10.2.3. Mac 平台下 Docker 安装

待补充。

10.3. Docker 私服构建

待补充。

10.4. 下载 Liberty 镜像

Liberty Docker 镜像官方网站 https://hub.docker.com/_/websphere-liberty/

通过 docker 命令可以直接下载镜像

```
docker pull websphere-liberty
```

10.5. 执行并运行应用程序

不加载应用，只运行最基本的镜像，并可以访问首页面

```
docker run -d -e LICENSE=accept -p 80:9080 websphere-liberty
```

加载应用并启动，然后访问应用

```
docker run -d -e LICENSE=accept -p 80:9080 -v  
/tmp/XyzApp.war:/opt/ibm/wlp/usr/servers/defaultServer/dropins/XyzApp.war websphere-liberty
```

10.6. 自定义镜像

创建 Dockerfile 文件，内容如下

```
FROM websphere-liberty:webProfile8  
COPY server.xml /config/  
COPY XyzApp.war /config/dropins/
```

构建镜像

```
docker build -t websphere-liberty/XyzApp:1.0.0 .
```

运行新镜像

```
docker run -d -e LICENSE=accept -p 80:9080 websphere-liberty/XyzApp:1.0.0
```

10.7. Docker 下 Liberty 集群构建

待补充。

11. 运行环境配置迷你化

由于 Liberty 基于 Java EE 规范而实现，但是在我们平时所开发的应用程序中并不需要所有的规范实现；因此，Liberty 在最终运行环境中可以进行定制化配置到最优最小，通过执行

```
server package <ServerName> --archive=<PackageFileName>.jar --include=minify
```

通过这种重新再打包然后进行再分发的方式，可以进一步降低了整个应用服务器的文件大小；测试了一下 JSP 与 Servlet 的应用，只需要 20 多兆即可以了。

详细的对比示例如下：

打包所有的 Liberty 服务器文件

```
server package defaultServer --archive=D:\Temp\LibertyAll.zip
```

仅最优最迷你化打包 Liberty 服务器文件：

```
server package defaultServer --archive=D:\Temp\LibertyMinify.zip --include=minify
```

然后比对 **LibertyMinify.zip** 文件和 **LibertyAll.zip** 文件的大小即可知道迷你化的效果。

其他功能：

如果文件名为 <PackageFileName>.jar 则是生成自解压文件，在 .jar 包中包含有 META-INF/MANIFEST.MF 文件，执行程序为 Main-Class: wlp.lib.extract.SelfExtract，因此通过执行

```
java -jar <PackageFileName>.jar
```

即可解压安装；

如果文件名是 <PackageFileName>.zip 则是单纯的压缩包，不包含自解压程序，于是可以通过 unzip 或 jar 进行解压

```
unzip <PackageFileName>.zip
```

进行解压即可。

通过 minify 进行迷你瘦身，它只打包应用程序所需要的功能组件包，并把不需要的功能组件包直接去除掉，从而达到最迷你。

12. 启用调试并进行远程调试

在启动 Liberty 时采用 `server debug` 进行启动即可启用调试模式，默认调试端口为 7777，如果想更改调试端口，可以在启动服务器前指定端口，比如

```
set WLP_DEBUG_ADDRESS=8888
```

或

```
export WLP_DEBUG_ADDRESS=8888
```

然后再启动，则可以看到

```
server debug
```

```
Listening for transport dt_socket at address: 8888
```

然后在 Eclipse 等 IDE 工具中进行配置，详见：

http://help.eclipse.org/kepler/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Fconcepts%2Fremdebug.htm&cp=1_2_12

13. 数据库连接池配置示例

启用功能特性组件

```
<featureManager>
  <feature>jdbc-4.2</feature>
  <feature>jndi-1.0</feature>
</featureManager>
```

13.1. DB2 数据库

```
<!-- DB2 -->
<library id="DB2JCC4Lib">
  <fileset dir="{wlp.user.dir}/shared/lib/" includes="db2jcc4.jar db2jcc_license_cisuz.jar"/>
</library>

<!-- DB2 NoneXA -->
<dataSource id="db2" jndiName="jdbc/db2" type="javax.sql.DataSource">
  <jdbcDriver libraryRef="DB2JCC4Lib"/>
  <properties.db2.jcc databaseName="tbankdb" serverName="localhost" portNumber="50000"
user="db2user" password="{aes}AIHtvA1aZOU6yB4W64u2BpvY29Pf3IkaLi+PxoxqZfle "
currentSchema="db2schema"/>
</dataSource>

<!-- DB2 XA -->
<dataSource id="db2xa" jndiName="jdbc/db2xa" type="javax.sql.XADataSource">
  <jdbcDriver libraryRef="DB2JCC4Lib"/>
  <properties.db2.jcc databaseName="tbankdb" serverName="localhost" portNumber="50000"
user="db2user" password="{aes}AIHtvA1aZOU6yB4W64u2BpvY29Pf3IkaLi+PxoxqZfle "
currentSchema="db2schema"/>
</dataSource>
```

13.2. Oracle 数据库

```
<!-- Oracle -->
<library id="OracleJar">
  <fileset dir="{wlp.user.dir}/shared/lib/" includes="ojdbc6.jar"/>
</library>
```



```

<jdbcDriver id="OracleJdbcDriver" libraryRef="OracleJar"/>
<dataSource          id="OracleDataSource"          jndiName="jdbc/OracleDataSource"
jdbcDriverRef="OracleJdbcDriver">
    <properties.oracle    URL="jdbc:oracle:thin:@192.168.1.100:1521:Orcl"    user="user1"
password="{aes} AIHtvA1aZOU6yB4W64u2BpvY29Pf3IkaLi+PxoxqZfle"/>
</dataSource>

```

13.3. MySQL 数据库

```

<!-- MySQL -->
<library id="MySQLLib">
    <file name="{wlp.user.dir}/shared/lib/mysql-connector-java-5.1.47.jar"/>
</library>
<jdbcDriver libraryRef="MySQLLib">
    <properties    databaseName="liberty"    serverName="localhost"    portNumber="3306"
user="liberty" password="{aes} AIHtvA1aZOU6yB4W64u2BpvY29Pf3IkaLi+PxoxqZfle"/>
</dataSource>
<dataSource id="DefaultDataSource" jndiName="jdbc/MySQL">

```

13.4. Derby 数据库

```

<!-- Derby Embedded -->
<library id="DerbyLib">
    <file name="D:\\OpenSource\\Derby\\10.14.1.0\\lib\\derby.jar"/>
</library>
<jdbcDriver id="DerbyEmbedded" libraryRef="DerbyLib"/>
<dataSource id="DefaultDataSource" jdbcDriverRef="DerbyEmbedded">
    <properties.derby.embedded databaseName="{server.config.dir}/database/derbyEmbedded"
createDatabase="create"/>
</dataSource>

<!-- Derby Network Client -->
<library id="DerbyLib">
    <fileset dir="D:\\OpenSource\\Derby\\10.14.1.0\\lib"/>
</library>
<jdbcDriver libraryRef="DerbyLib">
    <properties.derby.client databaseName="{server.config.dir}/database/derbyNetworkClient"

```

```
createDatabase="create" serverName="localhost" portNumber="1527"/>  
</dataSource>  
<dataSource id="DefaultDataSource" jndiName="jdbc/derbyClient">
```

更多可参见:

https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_dep_configuring_ds.html

14. 后记

Liberty 在开发时相当的方便与简单，但是要完整地构造一个可以投入正常运行的高可用性的、大型的生产环境，还是需要做不少的工作。

因此，在逐渐熟悉 Liberty 的过程积累下经验，并将可重用性的工具整理起来，为今后构建生产环境奠定基础。

此安装配置开发管理使用指南也仅是抛砖引玉之用，更多的内容请参考后面 更多的资料参考。

15. 资料参考

Liberty 主站点 : <http://wasdev.net>

Liberty 运行时与功能扩展包下载

<https://developer.ibm.com/wasdev/downloads/liberty-profile-using-non-eclipse-environments/>

Java EE 7 及 Java 8 新功能支持

<https://developer.ibm.com/wasdev/blog/2015/03/25/new-liberty-features-java-ee-7-plus-java-8-support/>

Liberty 系统列技术文章

<https://developer.ibm.com/wasdev/docs/category/articles/>

WebSphere Application Server Developer Tools for Eclipse:

http://publib.boulder.ibm.com/infocenter/radhelp/v9/topic/com.ibm.rad.install.doc/topics/t_install_wdt.html

WebSphere Application Server Liberty Profile Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/welc6tech_wlp_thr.html

Eclipse

<http://www.eclipse.org>

Jython

<http://www.jython.org>

Java2Class.net

<http://www.Java2Class.net>