# 目录

[目录 1](#_Toc381356816)

[版本修订 4](#_Toc381356817)

[1 背景 5](#_Toc381356818)

[2 目的 5](#_Toc381356819)

[3 内容 5](#_Toc381356820)

[4 OSGi 5](#_Toc381356821)

[4.1 OSGi Bundle简介 5](#_Toc381356822)

[5 集成开发环境配置 5](#_Toc381356823)

[5.1 开发环境简介 6](#_Toc381356824)

[5.2 准备一系列基础bundle 6](#_Toc381356825)

[5.3 建立Target Platform 7](#_Toc381356826)

[5.4 使用Eclipse开发Blueprint bundle 10](#_Toc381356827)

[5.5 在Eclipse中运行调试Bundle 16](#_Toc381356828)

[5.6 导出可部署的Bundle 17](#_Toc381356829)

[5.7 OSGi Bundle的部署与安装 19](#_Toc381356830)

[5.8 为blueprint应用在eclipse中提供xml内容提示 20](#_Toc381356831)

[5.9 OSGi控制台 20](#_Toc381356832)

[6 OSGi应用的部署和发布 22](#_Toc381356833)

[7 Apche Aries实践 31](#_Toc381356834)

[7.1 Apache aries blueprint最少需要哪些bundle支持 31](#_Toc381356835)

[7.2 Blueprint Bundle 31](#_Toc381356836)

[7.3 集成JPA 32](#_Toc381356837)

[7.3.1 JPA的概念 32](#_Toc381356838)

[7.3.2 JDBC 32](#_Toc381356839)

[7.3.2.1 OSGi如何支持JDBC 32](#_Toc381356840)

[7.3.2.2 OSGi JDBC Service 32](#_Toc381356841)

[7.3.2.3 Pax-JDBC 32](#_Toc381356842)

[7.3.3 JPA的主流实现 32](#_Toc381356843)

[7.3.3.1 Apache OpenJPA 33](#_Toc381356844)

[7.3.3.2 EclipseLink 33](#_Toc381356845)

[7.3.3.3 Hibernate 33](#_Toc381356846)

[7.3.3.4 Oracle TopLink 33](#_Toc381356847)

[7.3.4 Apache aries集成JPA需要哪些bundle 33](#_Toc381356848)

[7.3.4.1 Aries相关JPA Bundle 33](#_Toc381356849)

[7.3.4.2 相关JPA实现Bundle 33](#_Toc381356850)

[7.3.5 数据源配置 34](#_Toc381356851)

[7.3.5.1 Mysql 34](#_Toc381356852)

[7.3.5.2 Oracle 34](#_Toc381356853)

[7.3.5.3 DB2 34](#_Toc381356854)

[7.3.6 使用Apache Aries JPA 34](#_Toc381356855)

[7.3.6.1 设置数据源 35](#_Toc381356856)

[7.4 集成JNDI 35](#_Toc381356857)

[7.4.1 什么是JNDI 35](#_Toc381356858)

[7.4.2 Apache Aries集成JNDI 35](#_Toc381356859)

[7.4.2.1 如何配置JNDI服务 36](#_Toc381356860)

[7.4.3 通过JNDI获取Blueprint Container中的服务 36](#_Toc381356861)

[7.5 集成Transaction 37](#_Toc381356862)

[7.6 Aries application 37](#_Toc381356863)

[7.7 Enterprise Bundle Archive 38](#_Toc381356864)

[7.8 日志 39](#_Toc381356865)

[7.8.1 Log Service 39](#_Toc381356866)

[7.8.2 slf4j+logback 39](#_Toc381356867)

[7.8.3 Pax Logging 39](#_Toc381356868)

[7.8.3.1 如何使用pax logging 39](#_Toc381356869)

[8 OSGi开发最佳实践 40](#_Toc381356870)

[8.1 Bundle命名方法与规则 40](#_Toc381356871)

[8.2 模块划分 40](#_Toc381356872)

[8.3 依赖关系处理 40](#_Toc381356873)

[8.4 依赖于包而非bundle 40](#_Toc381356874)

[8.5 从实现中分离API 40](#_Toc381356875)

[8.6 尽可能应用OSGi提供的标准服务 40](#_Toc381356876)

[8.7 确定bundle的类型 41](#_Toc381356877)

[8.7.1 Fragment Bundle 41](#_Toc381356878)

[8.7.2 Utility Bundle 41](#_Toc381356879)

[8.7.3 构建Utility类型的Bundle 41](#_Toc381356880)

[8.8 Bundle的资源管理特性 42](#_Toc381356881)

[9 附录 43](#_Toc381356882)

[9.1 Apache Felix简介 43](#_Toc381356883)

[9.2 Apache Felix基本运行环境配置 43](#_Toc381356884)

[9.2.1 Apache Felix执行环境的下载 43](#_Toc381356885)

[9.2.2 启动Apache Felix框架 44](#_Toc381356886)

[9.3 Apache Felix基本控制命令 45](#_Toc381356887)

[9.4 在Eclipse中建立Felix开发环境 45](#_Toc381356888)

[9.5 Eclipse环境下运行bundle 50](#_Toc381356889)

[9.6 Apache Felix基本配置 51](#_Toc381356890)

[9.7 Web Console 51](#_Toc381356891)

[9.8 Shell 52](#_Toc381356892)

[9.9 File Install 53](#_Toc381356893)

[9.10 Lightweight Http 53](#_Toc381356894)

[9.11 BluePrint XML配置Gemini Blueprint与Blueprint的区别 53](#_Toc381356895)

[10 OSGi基础 55](#_Toc381356896)

[10.1 系统Bundle 55](#_Toc381356897)

# 版本修订

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 描述 | 修订人 |
| V1.0 | 2013-8-13 | 创建 | 陈绪绍 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 背景

基于WMS产品前期所定的有关OSGi的基本框架选型：Apache felix + Apache Aries。

# 目的

对当前团队来说，OSGi是一个较新的技术，涉及较为复杂的概念。另外，由于OSGi从4.0之后才逐步开始制定企业级应用相关的规范，相关的技术实现相对滞后，所以有必要对相关技术进行预研。以尽量减小后期开发风险。

# 内容

1. 研究与实践基于Eclipse平台进行OSGi开发所涉及的相关内容。包括开发环境搭建、开发步骤、调试方法、测试支持、打包部署等内容。

本预研文档，是一个迭代完善的过程。在后续的技术预研和实践当中，逐步添加与完善相关章节的内容。

# OSGi

## OSGi Bundle简介

# 集成开发环境配置

Eclipse本身提供了一个用于插件开发的工具(PDE)，只提供最基本的建立插件工程、运行、调试，导出等功能，对Equinox框架的支持最好。

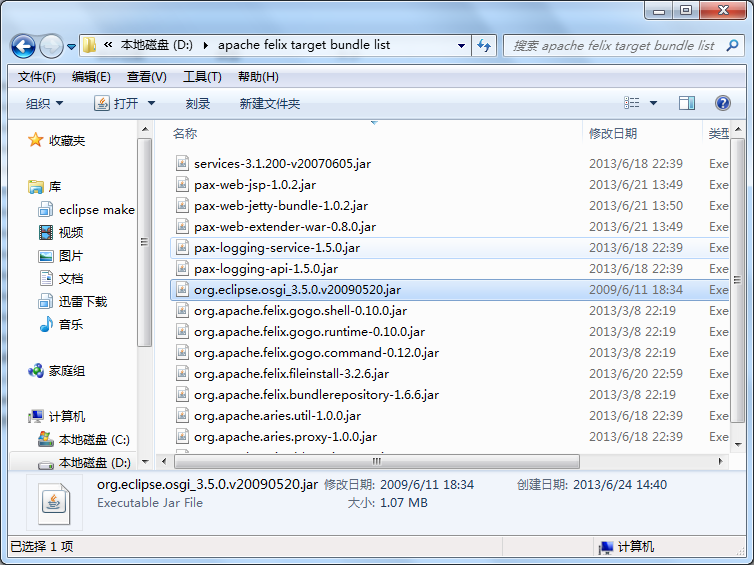
## 开发环境简介

您需要下载一个支持插件开发的Eclipse。安装 Eclipse 3.5或以上版本或Spring tool suit。

## 准备一系列基础bundle

为建立Target Platform作准备。准备一个目录，将所有基础Bundle拷贝至该目录。基础Bundle由项目组统一提供。

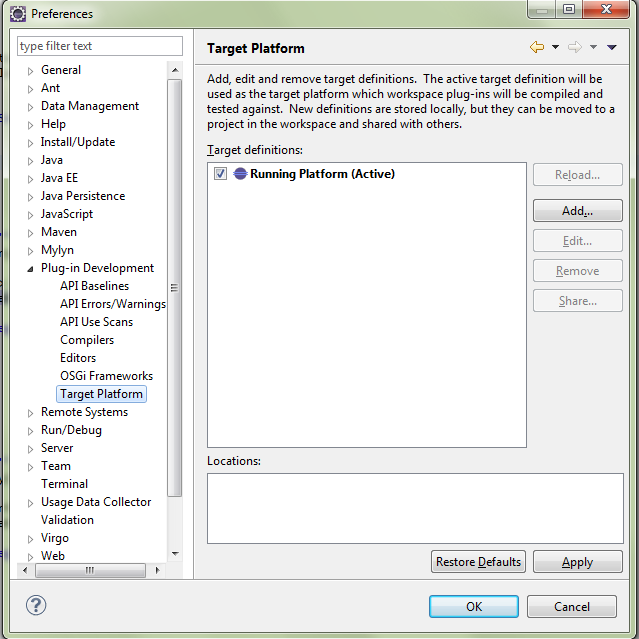
主要包括org.eclipse.osgi\*.jar(以Equinox作为内核，使用eclipse，只能以Equinox为内核)，最小的blueprint相关bundle，日志相关bundle，web支持的bundle，JNDI，JPA相关bundle。bundle列表一开始不要求是大而全的，在后期可以逐步加入。比如：



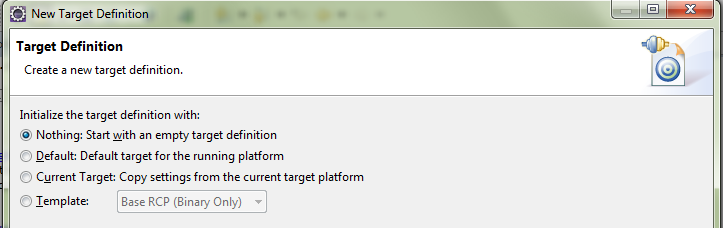
## 建立Target Platform

一句话快速上手：在Preferences🡪Plug-in Development🡪Target Platform中通过Add去建立新的目标平台。

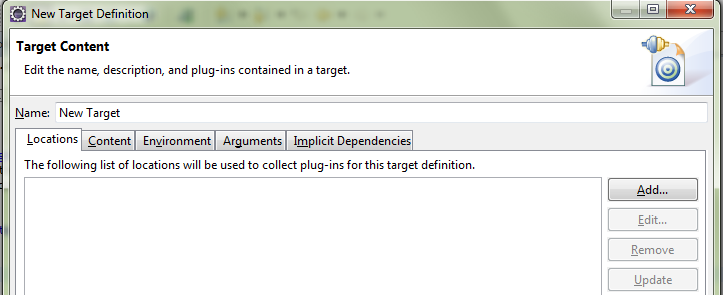
进入Eclipse的window->preferences界面，选中Plug-in Development下的Target Platform：



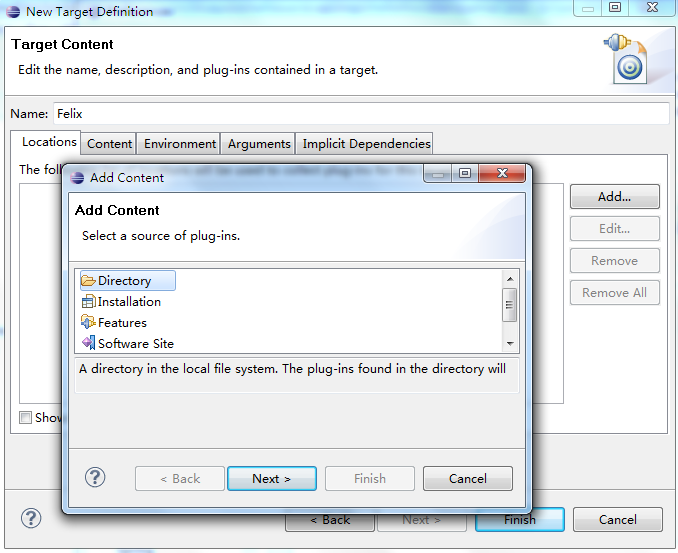
现在右边只有一个Runing Platform的，我们任务是点击“Add…”按钮来增加一个platform，进入如下界面：



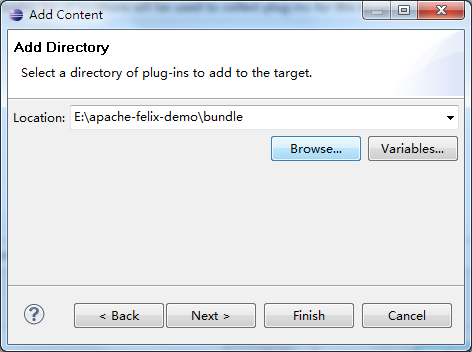
选择默认的第一个就好，点击next。



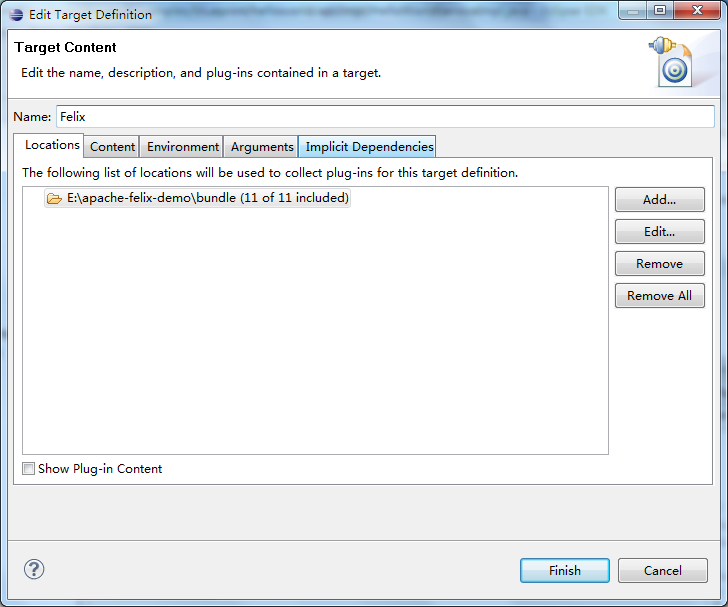
重命名为Felix(可以任意命名)。再点击这里的“Add…”：



选择“Directory”，选择准备的包含有你所准备好的bundle的目录。

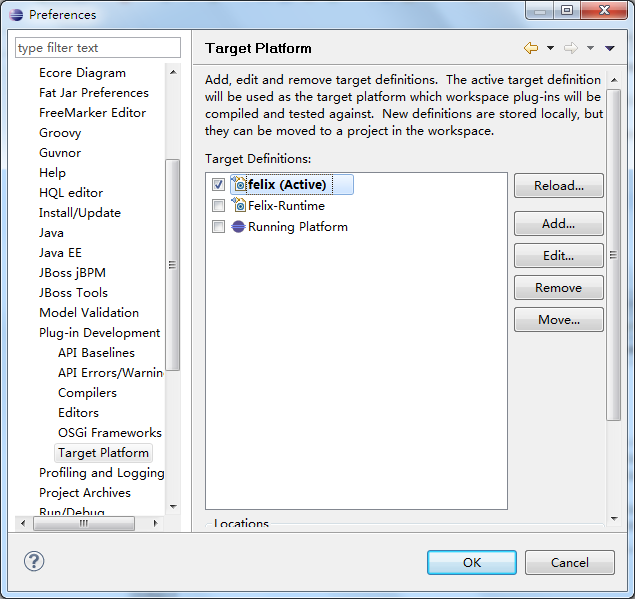


然后点击finish，回到刚才的界面：



这时候你就会发现里面多出来了你刚刚设置的路径，路径后面描述的“11 of 11 included”则就是说的我们放置的felix的jar包。

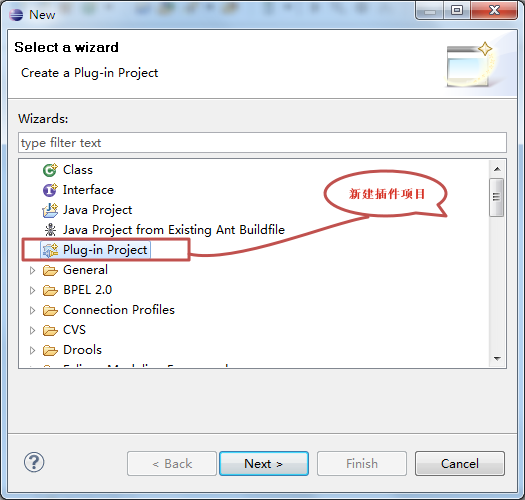
继续点击finish，回到最开始的界面：



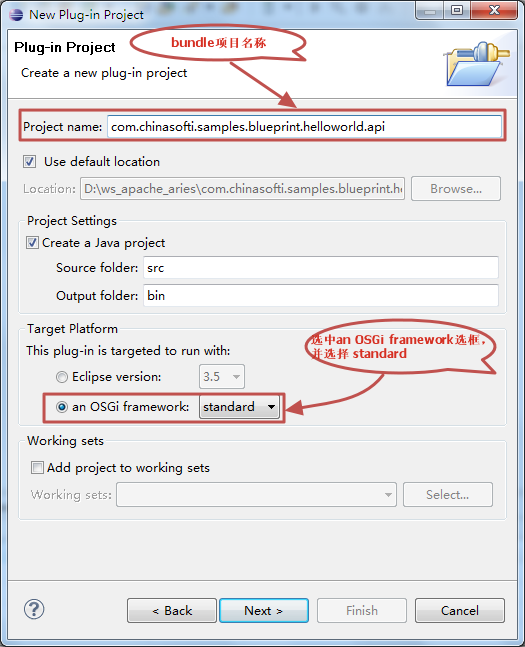
这时候多出来了一个新的target platform，即Feix，勾选上，然后确定。至此，基于Apache Aries的开发环境搭建完成。

## 使用Eclipse开发Blueprint bundle

1. 新建插件项目

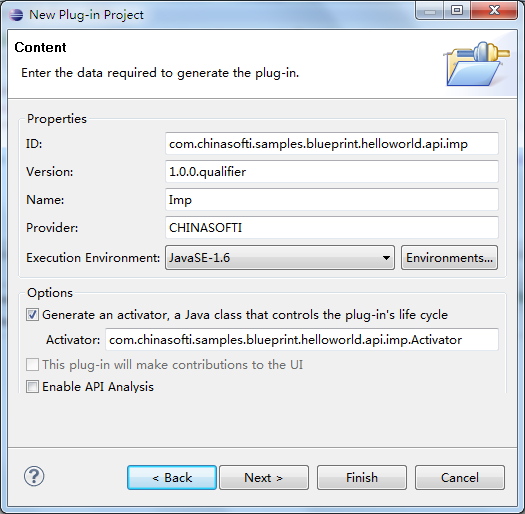


1. 选择an OSGi Framework

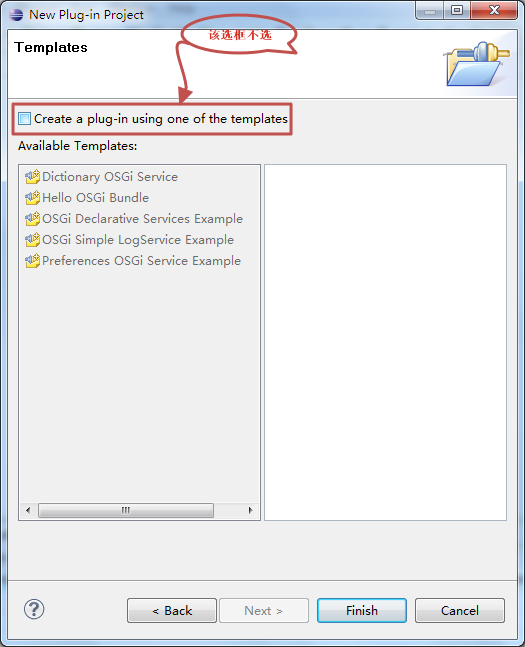


1. 建立纯净的OSGi Bundle工程模板

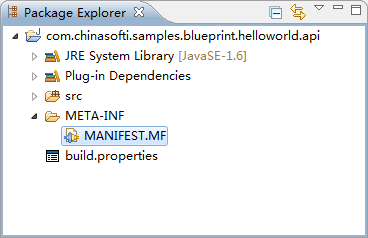
设置Bundle的相关信息。如ID，版本，提供者，运行环境及启动器等。



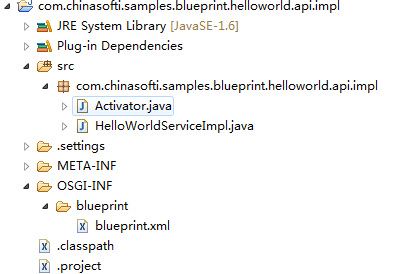
点击“下一步”



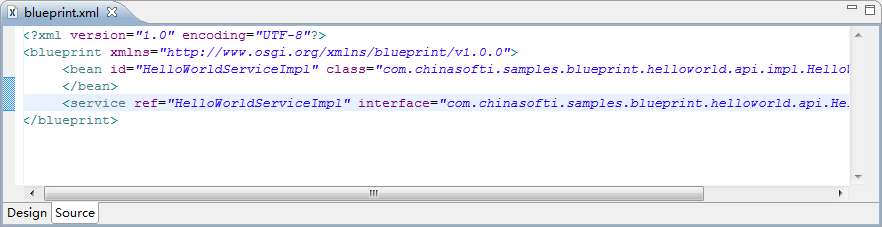
Bundle工程的基本目录结构



1. 工程根目录下建立OSGI-INF/blueprint目录，并在该目录下新建blueprint.xml文件

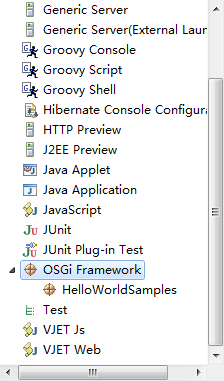


1. 编辑OSGI-INF/blueprint/blueprint.xml文件，进行bean注入，服务发布，引用等

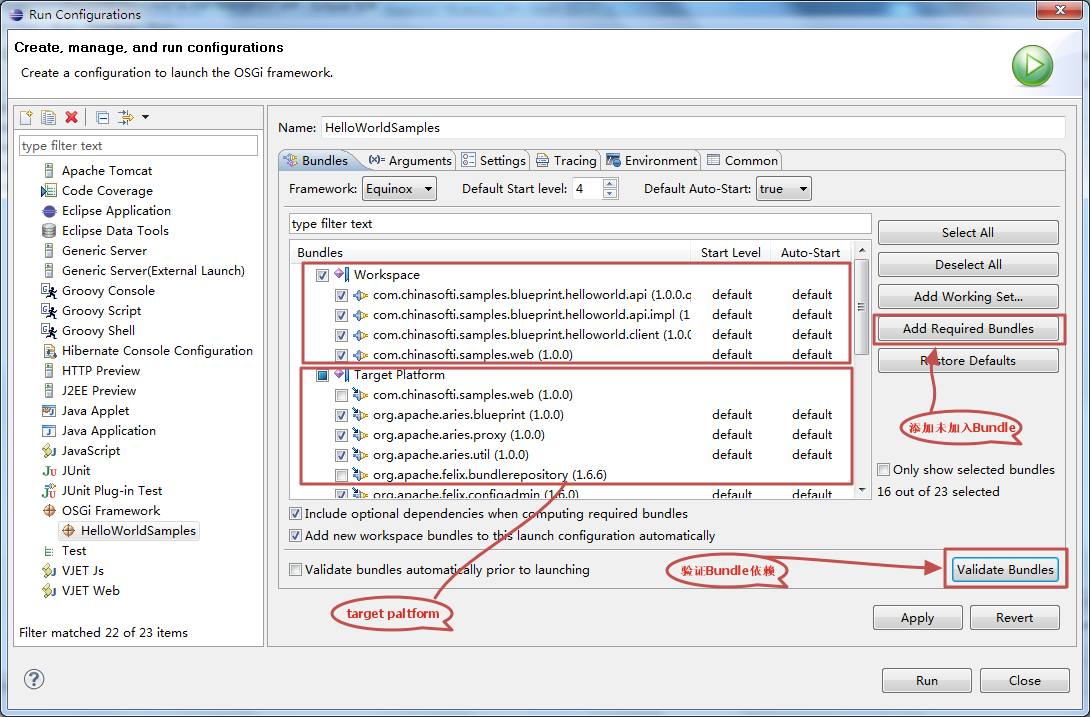


## 在Eclipse中运行调试Bundle

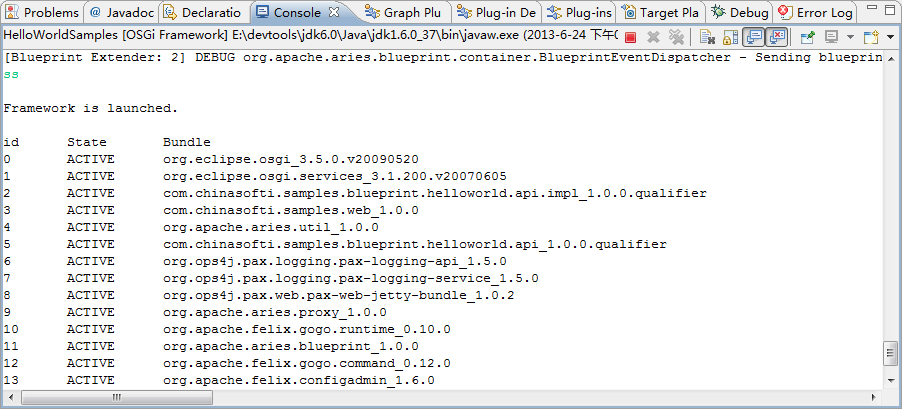
1. 运行：Run-->Run Configurations调试：Debug-->Debug Configurations
2. 双击点击OSGi Framework，或右键-->”New”。新建一个Run configuration节点。



弹出如下界面：

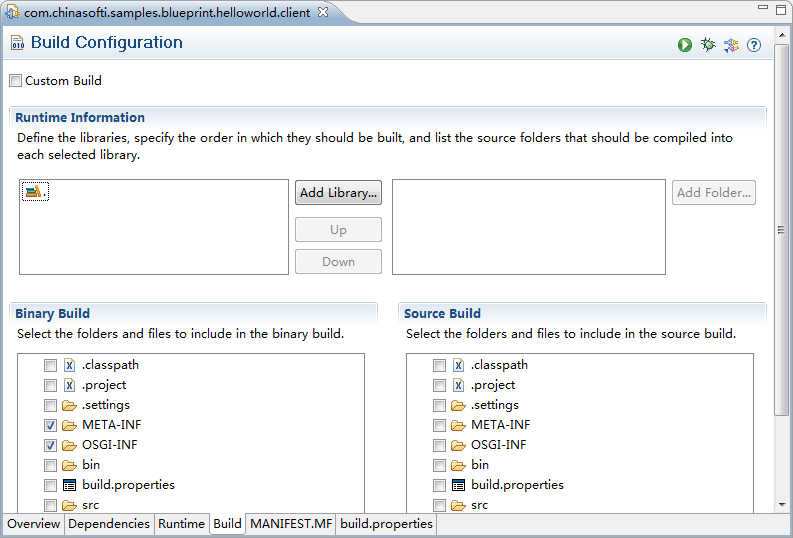


点击Run，即可在Eclipse平台运行或调试Bundle。



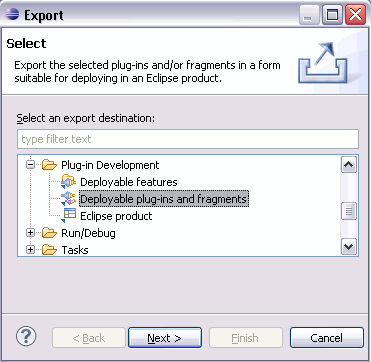
## 导出可部署的Bundle

打开bundle中的MANIFEST.MF文件,选择build页签，然后勾选要导出的内容。

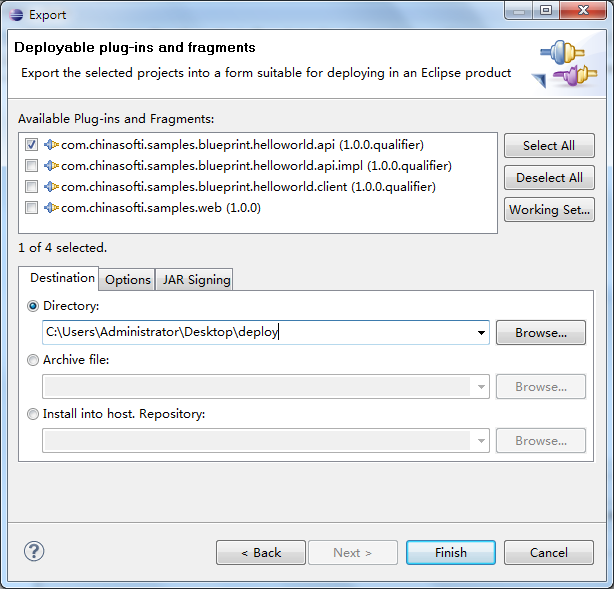


在build标签中选中将OSGI-INF导出，这一点很重要。

 Select your bundle and choose File -> Export -> Plug-in Development -> "Deployable plug-ins and fragment"。



选中Deployable plug-ins and fragment



勾选需要导出的bundle，点击finish即可。

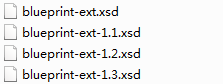
## OSGi Bundle的部署与安装

以Apache Felix框架为例。将导出的bundle拷贝至FELIX\_HOME\bundle目录下，启动Apache Felix框架，即完成了部署。Bundle的自动安装与更新，参见4.7章节。如果框架已经启动，则可通过gogo shell或web console来进行bundle的安装与部署。当然，如果你使用了fileinstall bundle，则不需要重启框架即可完成bundle的安装部署。

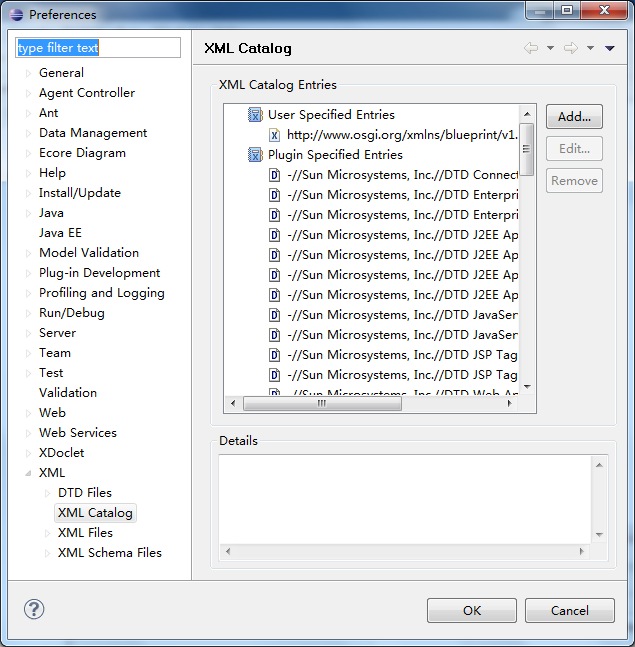
## 为blueprint应用在eclipse中提供xml内容提示

下载Apache aries源码，在如下目录下可以找到如下xsd文件。

pache-aries-src\_home\truck\blueprint\blueprint-core\src\main\resources\org\apache\aries\blueprint\ext\impl

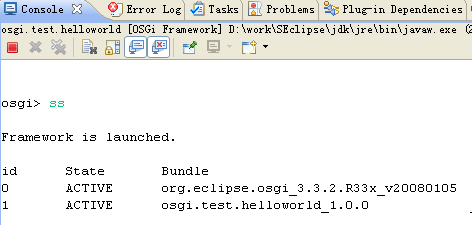


点击Preferences->XML->XML Catalog。



## OSGi控制台

OSGi 控制台对于习惯开发普通 Java 应用程序的开发人员来说，还是比较新鲜的。一般来说，通过 OSGi 控制台，您可以对系统中所有的 bundle 进行生命周期的管理，另外也可以查看系统环境，启动、停止整个框架，设置启动级别等等操作。如图，键入 SS 就可以查看所有 bundle 的状态：



下面列出了主要的控制台命令：

表 1. Equinox OSGi 主要的控制台命令表

|  |  |  |
| --- | --- | --- |
| **类别** | **命令** | **含义** |
| 控制框架 | launch | 启动框架 |
| shutdown | 停止框架 |
| close | 关闭、退出框架 |
| exit | 立即退出，相当于 System.exit |
| init | 卸载所有 bundle（前提是已经 shutdown） |
| setprop | 设置属性，在运行时进行 |
| 控制 bundle | Install | 安装 |
| uninstall | 卸载 |
| Start | 启动 |
| Stop | 停止 |
| Refresh | 刷新 |
| Update | 更新 |
| 展示状态 | Status | 展示安装的 bundle 和注册的服务 |
| Ss | 展示所有 bundle 的简单状态 |
| Services | 展示注册服务的详细信息 |
| Packages | 展示导入、导出包的状态 |
| Bundles | 展示所有已经安装的 bundles 的状态 |
| Headers | 展示 bundles 的头信息，即 MANIFEST.MF 中的内容 |
| Log | 展示 LOG 入口信息 |
| 其它 | Exec | 在另外一个进程中执行一个命令（阻塞状态） |
| Fork | 和 EXEC 不同的是不会引起阻塞 |
| Gc | 促使垃圾回收 |
| Getprop | 得到属性，或者某个属性 |
| 控制启动级别 | Sl | 得到某个 bundle 或者整个框架的 start level 信息 |
| Setfwsl | 设置框架的 start level |
| Setbsl | 设置 bundle 的 start level |
| setibsl | 设置初始化 bundle 的 start level |

# OSGi应用的部署和发布

部署osgi应用

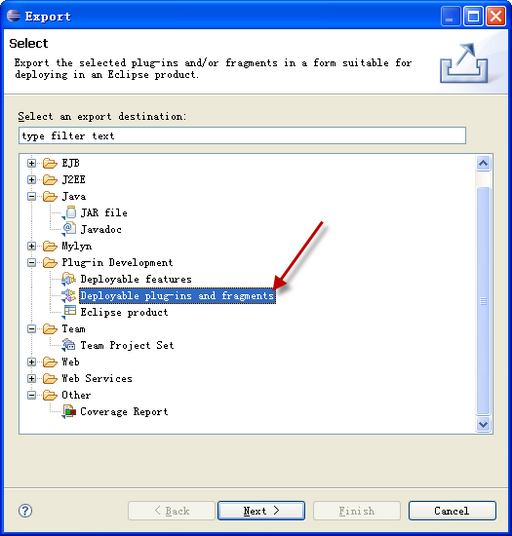
我们的 bundle 不会只能在 Eclipse 环境运行，我们需要能够将 bundle 部署到实际的操作系统中，可能是 Windows/Linux/Unix 等环境，这要求我们按照下列步骤进行：

发布 bundle，即将我们的 plug-in 工程发布为可以执行的 Jar 文件或者其它格式；

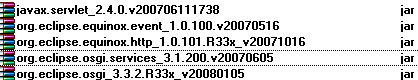
1. 配置 config.ini，指出 bundle 的运行环境，启动顺序等；
2. 启动脚本编写，编写能够运行在各种操作系统的脚本；

发布 Bundle

发布 bundle 的工作其实很简单，通过 eclipse 平台即可完成：

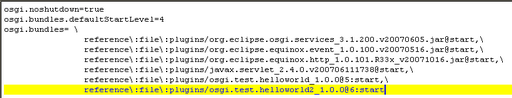
1. 选择 Eclipse 的 **plug-in** 视图的 File -> Export，从弹出的窗口中选择 **Deployable plug-ins and fragments**：   
   **选择 Deployable plug-ins and fragments**  
   
2. 在下一个窗口中，选择想要发布的 bundle，这里我们选择**osgi.test.helloworld**，**osgi.test.helloworld2** 工程，下面的 options 里面选择“打包为一个 Jar ”，目标目录选择为 osgi.test.deploy 目录（在当前的 workspace 下面）；
3. 选择确定，发布后的目录结构如下图所示，eclipse 帮我们在部署根目录下建立了一个新的子目录 plugins（类似于 eclipse，因为 eclipse 就是基于 OSGi 的）：   
     
   **图 30. 发布后的目录结构**  
   
4. 配置

为了让我们的 Jar 文件跑起来，需要 OSGi 的运行环境支持，所以我们需要拷贝一些 system bundle 到 plugins 目录中，包括：

**图 31. OSGi 的运行环境支持**  
 

然后，把 eclipse 目录的 org.eclipse.osgi\_3.3.2.R33x\_v20080105 文件拷贝到 osgi.test.deploy 根目录，重命名为 equinox.jar 文件。

在 osgi.test.deploy 目录新建子目录 configuration，新建一个文本文件 config.ini，用来配置 bundle 的启动环境，配置如下：

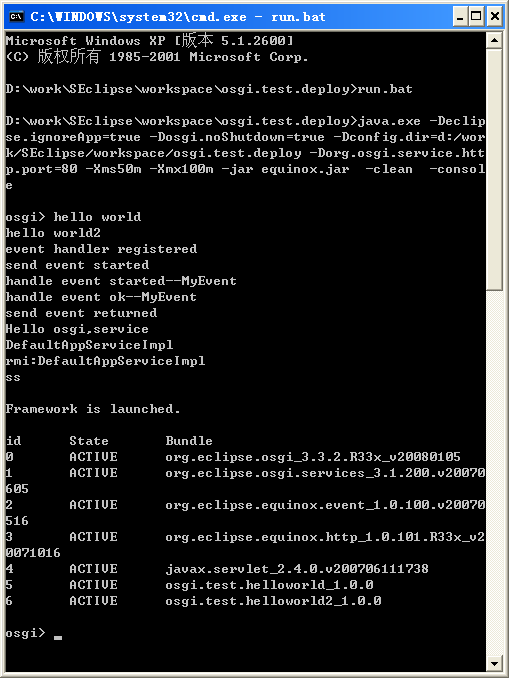
**图 32. config.ini 配置文件**  
 

注意最后两个 bundle 的启动顺序配置格式为：bundle@start\_leve:start。

好了，config.ini 也已经准备好了。

５.准备脚本

下面进行启动脚本编写，这个和普通的 Java 程序没有什么大的区别，都是调用 Java 程序执行一个 jar 文件，关键是其中的一些参数定义：

双击 run.bat，可以看到如下结果：  
  
**图 34. 运行结果**  


我们在OSGi控制台中运行我们的系统管理程序，其中还需要手工install, start各个Bundle,显然这很麻烦，不太象一个真正的应用程序，本节将介绍如何构造出一个完整的基于OSGi的应用程序。

首先，Equinox提供了在启动框架时自动安装Bundle以及启动Bundle的功能，这是通过定义config.ini文件来实现的，应用程序的目录结构如下：

|  |
| --- |
| E:\SYSMGT\_APP1  │ run.bat  │ org.eclipse.osgi\_3.2.0.v20060601.jar  ├─configuration  │ config.ini  └─plugins  com.systemmanagement.console\_1.0.0.jar  com.systemmanagement.cpumonitor\_1.0.0.jar  com.systemmanagement.services\_1.0.0.jar |

config.ini文件的内容如下：

|  |
| --- |
| osgi.bundles=plugins/com.systemmanagement.cpumonitor\_1.0.0.jar@1:start,  plugins/com.systemmanagement.console\_1.0.0.jar@2:start,  plugins/com.systemmanagement.services\_1.0.0.jar |

其中的@1,@2用于指定Bundle的Start Level, start表示当OSGi框架启动后即自动启动此Bundle。而run.bat是一个批处理程序，其内容如下：

java –jar org.eclipse.osgi\_3.2.0.v20060601.jar -console

现在只需运行run.bat即可运行我们的系统管理应用程序了。

Eclipse中还提供了两个Bundle, org.eclipse.equinox.common和org.eclipse.update.configurator，可用于自动发现和安装指定目录下新增加的Bundle。我们同样可以在Eclipse的plugins目录中找到这两个Bundle并将它们拷到E:\SYSMGT\_APP1\plugins目录下，并修改config.ini文件的内容如下所示：

|  |
| --- |
| osgi.bundles=plugins/org.eclipse.equinox.common\_3.2.0.v20060603.jar@1:start,  plugins/org.eclipse.update.configurator\_3.2.0.v20060605.jar@2:start,  plugins/com.systemmanagement.cpumonitor\_1.0.0.jar@2:start,  plugins/com.systemmanagement.console\_1.0.0.jar@3:start |

这样，当org.eclipse.update.configurator Bundle启动时，它会去自动发现和安装plugins目录新增的Bundle，但需注意，它并不能自动启动这些Bundle，OSGi协议目前也没有提供这样的标准服务来自动安装，管理和启动Bundle，我们可以使用如下的一些解决方案来更好地管理和分发Bundle:

* 利用Eclipse的Update Manager：Eclipse Update Manager提供了一批API用于管理，下载，升级，安装Feature，一个Feature是一批Bundle的集合, 是下载和安装的最小单位。但Feature是Eclipse定义的，不是OSGi协议中定义的特性。
* 使用FileInstall Bundle,这个Bundle可以监视某一个目录，如果这个目录中新增加了Bundle,它会自动将其install,如果有Bundle被更新了，它会自动将其update，如果有Bundle从此目录被移走，它会自动将其uninstall。可到<http://www.aqute.biz/Code/FileInstall>下载这个Bundle，它可以运行于任何OSGi框架中。

安装bundle

eg:

install reference:file:F:\workspace\_osgi\HelloWorldOsgi\

或

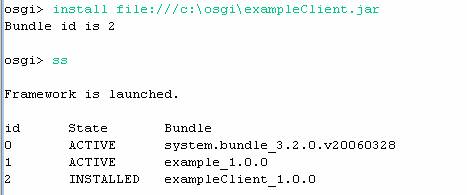
install file:F:\workspace\_osgi\HelloWorldOsgi\

★bundle同样可以远程安装，启动Bundle

eg:

install<http://www.eclipsezone.com/files/jsig/bundles/HelloWorld.jar>

eg:

**安装Bundle**  


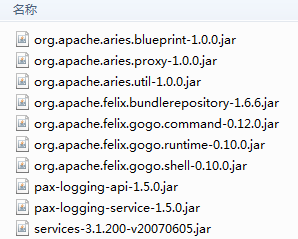
# Apche Aries实践

## Apache aries blueprint最少需要哪些bundle支持

基于apache felix环境，需要jdk6.0，否则报错

* 最基本的felix bundle，即通过官方下载的distribution包。Gogo shell之类的不是必需的，完全可以使用其它的shell，比如Equinox的。
* org.apache.aries.blueprint-1.0.0.jar [依赖slf日志组件，aries.util bundle]
* org.apache.aries.proxy-1.0.0.jar [依赖slf日志组件]
* org.apache.aries.util-1.0.0.jar
* pax-logging-service-1.5.0.jar [来自pax logging项目]
* pax-logging-api-1.5.0.jar [来自pax logging项目]
* services-3.1.200-v20070605.jar [对osgi基础服务的实现bundle，取自equinox的实现，即org.eclipse.osgi.services\_version.jar，也可以从osgi官方去下载]

如下图所示：



## Blueprint Bundle

对一个OSGi Bundle而言，在/OSGI-INF/blueprint 路径下有\*.xml配置文件，该bundle就可以被称之为 Blueprint Bundle，它可以被 Blueprint Container 发现并进行相关的操作。

* 默认的配置文件为/OSGI-INF/blueprint 路径下的 blueprint.xml 文件。
* 非默认路径下的 blueprint.xml 文件，在指定这个位置的 MANIFEST.MF 中附有一个 Bundle-Blueprint 头部。

## 集成JPA

### JPA的概念

JPA(JAVA Persistence API)是sun官方提出的Java持久化规范。他为Java开发人员提供了一种对象/关系映射工具来管理Java应用中的关系数据库。他的出现主要是为了简化现有的持久化开发工作和整ORM技术，结束现在Hibernate,TopLink,JDO等ORM框架各自为营的局面。

### JDBC

JDBC（Java Data Base Connectivity,java数据库连接），由一些接口和类构成的API。由java.sql,javax.sql包组成。

#### OSGi如何支持JDBC

#### OSGi JDBC Service

#### Pax-JDBC

### JPA的主流实现

JPA 的目标之一是制定一个可以由很多供应商实现的API，目前OpenJpa、EclipseLink、Hibernate、TopLink等都提供了JPA的实现

#### Apache OpenJPA

OpenJPA是Apache提供的开源Java持久化框架。

#### EclipseLink

其前身是Oracle的TopLink[ORM部分]。官方专门提供了EclipseLink对应的OSGi Bundle。



#### Hibernate

Hibernate 从3.2开始兼容JPA。

#### Oracle TopLink

由Oracle提供的JPA实现框架。TopLink与EclipseLink不同在于，以后的TopLink将在EclipseLink基础上增加与Oracle其他产品整合的代码，TopLink有自己的Oracle TopLink Workbench，这个Oracle TopLink Workbench并没有捐献给开源社区。

### Apache aries集成JPA需要哪些bundle

#### Aries相关JPA Bundle

org.apache.aries.jpa.api-1.0.0.jar

org.apache.aries.jpa.container.context-1.0.0.jar

#### 相关JPA实现Bundle

不同的JPA提供商所提供的JPA需要不同的bundle支持。

1. OpenJPA
2. EclipseLink

### 数据源配置

#### Mysql

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  <beanid="dataSource" class="com.mysql.jdbc.jdbc2.optional.MysqlDataSource">  <property name="url" value="jdbc:mysql://localhost:3306/world"/>  <property name="user" value="test"/>  <property name="password" value="test"/>  </bean>  <service interface="javax.sql.DataSource" ref="dataSource">  <service-properties>  <entry key="osgi.jndi.service.name" value="jdbc/mysqlds"/>  </service-properties>  </service>  </blueprint> |

#### Oracle

#### DB2

### 使用Apache Aries JPA

1. 新建JPA配置文件

在bundle根目录下新建META-INF/persistence.xml文件。

META-INF/MANIFEST.MF文件中可能要增加的头部：

* **Meta-Persistence：**这个头部的存在表明这是一个持久性捆绑包，可用于指定 persistence.xml 文件的位置（如果不在默认 /META-INF 路径中）。
* **Export-Package：**在 JAR 实用程序中包含的包的列表。在包的末尾显示的版本是可选的，分配一个版本号给导出的包作为这个捆绑包的一部分。
* **Import-Package：**捆绑包中的类所需要的（作为导入）包的列表。不同应用程序中有所不同。包末尾的版本号是可选的，将指定能够接受的最小版本号或版本范围，导入到捆绑包中。

任何空白头部，比如上面提到的 Meta-Persistence，必须在冒号之后包含一个空格。除此之外，在一行的末尾或者最后一项下面的行中（在最后一项的末尾的回车之后）不能有额外 “空格”。

更多JPA内容参考：<http://aries.apache.org/modules/jpaproject.html>

#### 设置数据源

## 集成JNDI

### 什么是JNDI

### Apache Aries集成JNDI

Apache aries集成JNDI最少需要如下两个bundle:

org.apache.aries.jndi-1.0.0.jar

org.apache.aries.jndi.url-1.0.0.jar

作为注释这个 bean Java 代码的一种替代方法，容器托管的 PersistenceContext 能够通过在 Blueprint bean 定义中使用 <jpa:context> 元素来声明，如清单 9 所示。

**清单 9. Blueprint 容器托管的 Transaction 和 Persistence 配置**

|  |
| --- |
| <bean id="persistenceManager"  class="com.ibm.ws.eba.example.blog.persistence.BlogPersistenceServiceImpl">  <tx:transaction method="\*" value="Required"/>  <jpa:context property="em" unitname="blogExample"/>  </bean> |

清单 9 还展示了一些 Blueprint 容器托管的事务。在这个示例中，persistenceManager bean 的所有方法都在 Blueprint 容器建立（或连接）的一个全局事务下运行。

#### 如何配置JNDI服务

如：

<bean id="derbyDataSource" class="org.apache.derby.jdbc.EmbeddedDataSource">

<property name="databaseName" value="memory:blogDB"/>

</bean>

<service ref="derbyDataSource" interface="javax.sql.DataSource">

<service-properties>

<entry key="osgi.jndi.service.name" value="jdbc/blogdbnojta"/>

</service-properties>

</service>

### 通过JNDI获取Blueprint Container中的服务

1. 发布服务

|  |
| --- |
| <blueprintxmlns=...*>*  <beanid=*"bloggingServiceComponent"* class=*"org.apache.aries.BloggingServiceImpl"*>  </bean>  <serviceref=*"bloggingServiceComponent"* interface=*"org.apache.aries.samples.blog.api.BloggingService"*/>  ...  </blueprint*>* |

1. 获取服务

|  |
| --- |
| InitialContext ic = **new** InitialContext();  BloggingService blog= ic.lookup("osgi:services/"  + BloggingService.**class**.getName()); |

所有 OSGi 服务都自动在 JNDI 中注册，并可以以访问 Java EE 组件的熟悉方式访问

**从 Web 组件访问 OSGi 服务**

|  |
| --- |
| InitialContext ic = new InitialContext();  return (BloggingService) ic.lookup("osgi:service/"  + BloggingService.class.getName()); |

使用场景，比如在HttpServet中去获取相应的Service，处理相应的业务逻辑。

## 集成Transaction

## Aries application

一个Aries应用程序是一个或多个OSGI模块的集合，这些模块组合在一起提供相关的业务功能。一个Aries应用程序可以有多个模块以不同的方式组合构成。例如，一个Aries应用程序提供web banking功能，这个功能可以由一些web内容（web application bundles）、blueprint上下文bundle、(JPA实体bundle+持久化配置文件)bundle组成。

一个Aries应用程序与他自身模块所提供的服务隔离，这样做的目的是应用程序不为外界所见，除非清晰地配置导出功能。一个aries应用程序有多种方式接受工作负载（用户请求）：一个aries应用程序可以包括web bundle来执行http的访问，也可以导出一个个的service为其他aries应用程序所用或者发布成web Service。

在隔离的运行环境中，每个Aries应用程序与他的OSGI服务注册库一起，运行在它自身隔离的OSGI框架实例中。一个aries应用程序中的bundles，不能看到另一个OSGI应用程序中定义的bundle,service或者package,除非这些bundles,services ,packages被清晰地被指名应用程序共享。一种Aries应用程序可以装在和消费来自共享bundle空间的OSGI服务。也就是说，这种OSGI框架实例是所有隔离的OSGI应用程序框架实例的父类。（只有暴露，才能被共享）

一个Aries应用程序被打包成zip文件，并且以.eba为后缀。

## Enterprise Bundle Archive

作为 Aries 编程模型的一部分，Enterprise Bundle Archive 组件模型扩展了 Blueprint 组件模型，用于将有内在业务关联的 OSGi 模块打包在一起构成一个 OSGi 应用，并与其它 OSGi 应用相隔离。EBA 中既可以包含 Blueprint Bundle，也可以包含 Web 应用 Bundle 或者持久化 Bundle，此外，EBA 中还会附加一些应用的元数据。在某种程度上，EBA 很类似于传统 JavaEE 应用中的 EAR，所不同的是，EBA 中可以不直接包含 OSGi Bundle 的二进制内容，而是引用位于 OSGi Bundle 仓库中的 Bundle，这样 EBA 中的 Web 模块就可以引用位于集中式的，受管的，版本化的 Bundle 仓库中的共享类库和服务，所带来的是资源的节省和应用升级的便利。

EBA中的Manifest 文件 Application.MF 等价于 Bundle 中的 MANIFEST.MF，将模块化定义扩展到了应用级别上，如下所示。

**Application.MF**

|  |
| --- |
| Manifest-Version: 1.0  Application-Name: WeatherForecastApp  Application-SymbolicName: com.ibm.dw.weatherforecast.app  Application-Version: 1.0  Application-ManifestVersion: 1.0  Application-Content: com.ibm.dw.weatherforecast.api;version=1.0.0,  com.ibm.dw.weatherforecast.service;version=1.0.0,  com.ibm.dw.weatherforecast.web;version=1.0.0 |

## 日志

由于OSGi类加载器的原因，log4j等传统java应用中经常使用的日志组件在OSGi环境中是不能直接使用的。能够使用的日志有，OSGi Log Service，slf+logback，pax logging。

### Log Service

OSGi提供的日志服务。不同框架均有实现。不太好用。使用日志接口前需要获取日志服务。如:

|  |
| --- |
| //获取OSGi Log服务的服务引用  ServiceReference logSr = context.getServiceReference(LogService.class.getName()); //根据Log服务引用获取LogService服务实例  log = (LogService) context.getService(logSr);  log.log(LogService.LOG\_INFO, “log content”); |

### slf4j+logback

需要自己下载相应的日志包，然后将其包装为bundle。

### Pax Logging

由OPS4j提供的日志组件，支持log4j，slf,commons logging，jdk logger等，可以像使用传统项目中那样写日志。目前Apache Aries也是基于pax logging。

#### 如何使用pax logging

使用pax logging需要以下bundle支持

pax-confman-propsloader-0.2.2.jar [提供一个日志配置服务]

org.apache.felix.configadmin-1.6.0.jar

然后在felix\_home目录下建立如下目录

configurations/services/

该目录下放置日志配置文件：

org.ops4j.pax.logging.properties

# OSGi开发最佳实践

## Bundle命名方法与规则

符号名（symbolic name）的值遵循与Java包命名相似的规则：由一系列点分（dot-separated）字符串组成，推荐使用反向域名命名从而避免名字冲突。

## 模块划分

## 依赖关系处理

## 依赖于包而非bundle

## 从实现中分离API

## ****尽可能应用OSGi提供的标准服务****

OSGi联盟为一些经常用到的功能定义了标准服务，如应用程序管理服务，日志服务，事件服务，配置管理服务，用户管理服务，HTTP服务等等。如果开发人员采用的OSGi环境提供了上述服务的实现，且这些服务实现满足系统的需求，则开发人员应尽可能使用这些服务，而不是为这些类似的功能定义自己的实现。

## 确定bundle的类型

### Fragment Bundle

### Utility Bundle

这种Bundle与通常开发方式中的工具类或工具类包没有本质的区别，仅仅是将这些工具类发布到OSGi环境中，这些工具类也不依赖任何的OSGi环境或特性。通常，这种Bundle特别适合目前存在的众多的第三方组件引入到OSGi环境中直接供OSGi开发人员使用。例如，我们可以将Apache开源项目发布的的Commons工具包通过修改其META-INF目录下的MANIFEST.MF文件，添加Bundle的元数据信息就可以直接将其封装为可供OSGi环境中其他Bundle使用的工具类Bundle。

### ****构建Utility类型的Bundle****

该示例将第三方log4j工具包（版本为1.2.13）封装为OSGi Bundle组件。操作步骤如下：

步骤一：在C盘根目录下创建名称为"org.apache.log4j"目录，将log4j.1.2.13.jar文件拷贝到该目录中；

步骤二：在"org.apache.log4j"目录下构建META-INF目录，并在此目录下创建名称为"MANIFEST.MF"文件；

步骤三：编辑"MANIFEST.MF"文件，在此文件中添加以下内容：

Manifest-Version: 1.0   
Bundle-ManifestVersion: 2   
Bundle-Name: Apache Log4j   
Bundle-SymbolicName: org.apache.log4j   
Bundle-Version: 1.2.13   
Bundle-ClassPath: .;log4j.1.2.13.jar   
Export-Package: org.apache.log4j;version="1.2.13",   
org.apache.log4j.config;version="1.2.13",   
org.apache.log4j.helpers;version="1.2.13",   
org.apache.log4j.jdbc;version="1.2.13",   
org.apache.log4j.jmx;version="1.2.13",   
org.apache.log4j.net;version="1.2.13",   
org.apache.log4j.or;version="1.2.13",   
org.apache.log4j.or.jms;version="1.2.13",   
org.apache.log4j.or.sax;version="1.2.13",   
org.apache.log4j.spi;version="1.2.13",   
org.apache.log4j.varia;version="1.2.13",   
org.apache.log4j.xml;version="1.2.13"

经过上述步骤后，名称为org.apache.log4j的OSGi Bundle已经构建完成。开发人员可以在其他Bundle中如同平常的开发模式一样引用Log4j的类包，获取Logger实例记录日志。

## ****Bundle的资源管理特性****

在通常的应用系统开发中，开发人员通常要考虑配置文件的存储路径，系统临时文件的存储路径等信息。由于OSGi环境中Bundle实际上是由一组文件资源构成，开发人员在开发Bundle时可以充分利用这种特点，使用Bundle存储与Bundle相关的配置信息。OSGi环境在运行时为每一个Bundle构建一个序列化数据存储区，开发人员可以使用该存储区存储Bundle在运行时生成的临时信息。

下述代码片段展示了如何通过Bundle提供的功能接口获取Bundle内部的资源：

context.getBundle.getEntry("")只查询Bundle内部资源，返回Bundle的URL，该URL与实现相关，如：<bundleentry://256/>

context.getBundle.getEntry("/")只查询Bundle内部资源，返回Bundle的URL，该URL与实现相关

context.getBundle.getEntry("/build.properties")只查询Bundle内部资源，返回Bundle内部build.properties文件的URL，该URL与实现相关，如：<bundleentry://256/build.properties>

context.getBundle().getResource("build.properties")使用Bundle的ClassLoader查询Bundle类域内的资源，如果Bundle不能被解析（RESOLVED），则只查询Bundle内部资源，返回Bundle内部build.properties文件的URL，该URL与实现相关。

开发人员在获取上述资源的URL后，可以转换成与Bundle运行系统相关的文件URL，如：file:C:/test\_bundle\_1.0.0。

下述代码片段展示了通过BundleContext接口获取Bundle运行时数据存储区资源：

context.getDataFile("")返回该Bundle的运行时数据存储区位置，如：E:/worksapce/.metadata/.plugins/org.eclipse.pde.core/CobWeb/org.eclipse.osgi/bundles/256/data

context.getDataFile("config.xml")返回该Bundle的运行时数据存储区内的config.xml文件，如果该文件不存在，则创建该文件。

# 附录

## Apache Felix简介

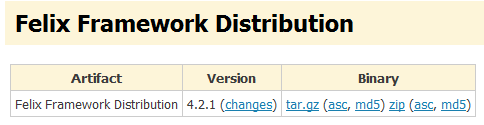
Apace Felix是Apache Foundation关于OSGi R4的一个实现。包括了OSGi框架和标准的服务，同时也提供并且支持其他的OSGi相关技术。Felix最终的目标是要提供和OSGi框架和标准服务完全兼容的一个实现。目前Felix已经实现了OSGi R4规范中的大部分内容，具体可以参考http://felix.apache.org/site/index.html

## Apache Felix基本运行环境配置

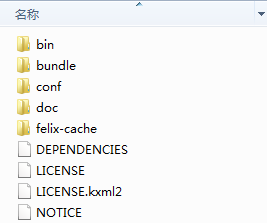
### Apache Felix执行环境的下载

进入下载页面，选择Felix Framework Distribution的下载项：

<http://felix.apache.org/site/downloads.cgi>。如下图所示：



Windows用户下载zip格式，Linux用户下载tar.gz格式。下载完成后，解压，得到如下图所示的目录结构。

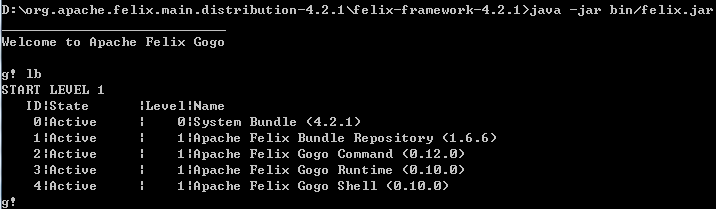


### 启动Apache Felix框架

进入felix根目录，执行如下命令，进入到Felix的命令行：

java-jar bin/felix.jar

如下图所示：



这样我们就进入了Felix的控制台，可以通过Shell与框架进行一些交互。

## Apache Felix基本控制命令

1. lb：列举所有bundle，支持过滤
2. bundles：同lb
3. install 安装bundle

eg:install file:bundle/listener.jar or start file:/c:/tutorial/example1.jar

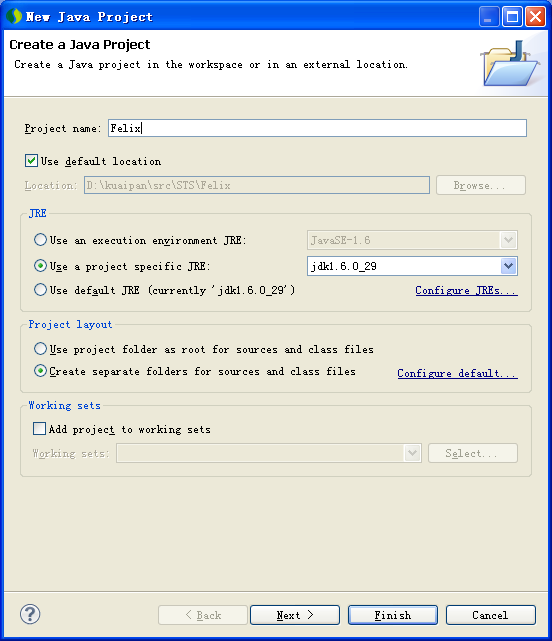
1. start 启动bundlestart 7
2. stop 停止bundle
3. uninstall 卸载bundle eg:uninstall file:bundle/listener.jar

## 在Eclipse中建立Felix开发环境

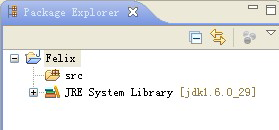
先下载Apache Felix并解压至某目录。参考7.2.1。

接下来我们通过Eclipse来配置Apache Felix

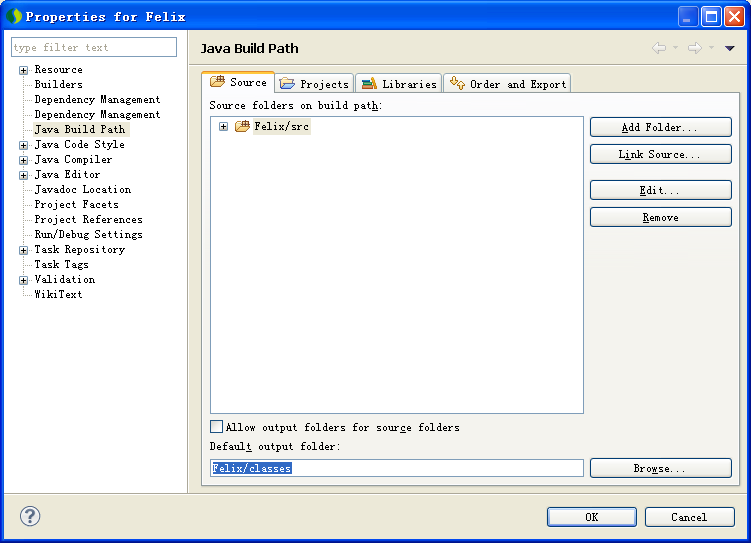
1. 创建一个新的Java工程：



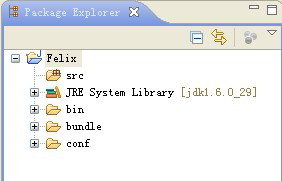
完成后的工程如下图所示：



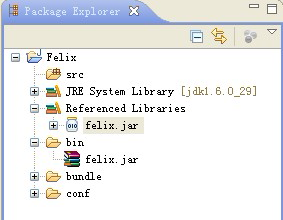
1. 修改默认Output文件夹，选中Felix工程，Alt+Enter快捷键，将输出文件夹改为Felix/classes，单击OK按钮。



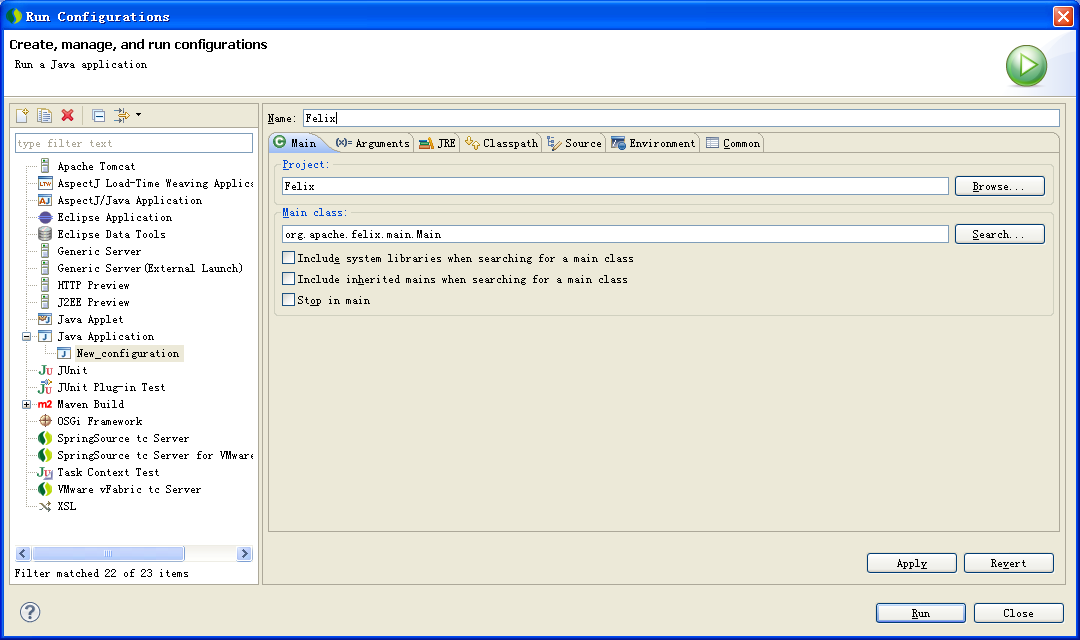
1. 复制 Felix Framework Distribution 文件。将刚才解压的Felix Framework Distribution的bin、bundle、conf文件复制到新创建的Felix的工程目录中，完成后的工程显示如下图所示。



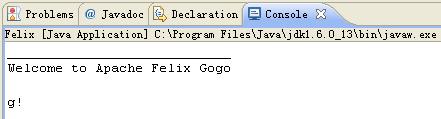
1. 将bin文件中felix.jar加入到Build path中，右击Felix.jar-->Build Path-->Add to Build Path，完成后的工程显示如下图所示。



1. 配置Run Configurations



单击Run按钮，就可以启动Felix了，运行的结果如下图所示。



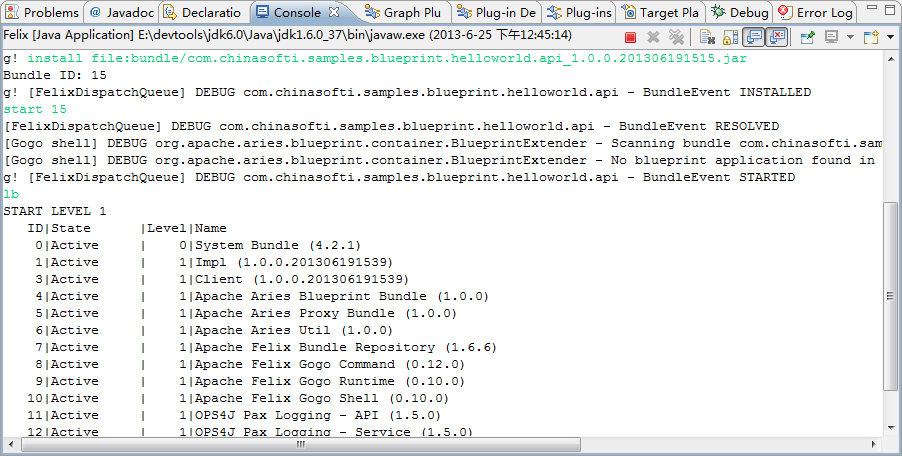
## Eclipse环境下运行bundle

1. 新建一个插件工程并导出为可部署的bundle。参考4章节内容。
2. 接着就是安装、运行了。

有三种方法可以安装、运行一个bundle。

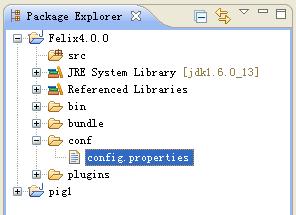
1. 使用命令。

首先，启动Felix，在Console中先使用install命令安装bundle，接着使用start命令启动bundle，如下图：



启动的时候，start命令后接着那个bundle的启动ID就可以启动bundle了，如上图的15。当然，你也可以使用uninstall命令卸载一个bundle，用法如install命令。

1. 使用Felix配置文件，打开conf/config.properties，如下图：



打开config.properties，找到felix.auto.start.1参数，值写成

felix.auto.start.1=bundle/com.chinasofti.samples.blueprint.helloworld.api\_1.0.0.201306191515.jar

多个bundle，之间用空格隔开。

参数写好后，启动Felix，你就可以看到bundle自动安装并启动了

1. 第三种方法就是使用File Install了，使用Apache Felix的File Install bundle，我们可以安装和启动bundle而无需启动Felix，参见7.9章节。

## Apache Felix基本配置

配置文件位于FELIX\_HOME\conf\config.properties

* org.osgi.framework.storage.clean - 决定bundle缓存是否刷新。值是“none”或“onFirstInit”。“none”不刷新缓存；“onFirstInit”当框架首次初始化时刷新缓存。默认值是“none”。
* org.osgi.service.http.port=8181–配置http服务端口
* felix.fileinstall.dir=deploy安装fileinstall服务后对监控目录进行配置

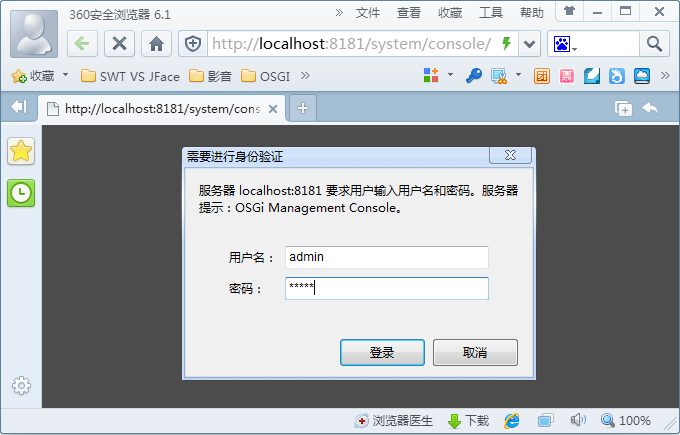
## Web Console

Web Console可以对OSGi环境进行如下一系列操作，包括bundle的安装，启动，停止，更新，日志查看等。实际上Web Console也是Shell的一种。

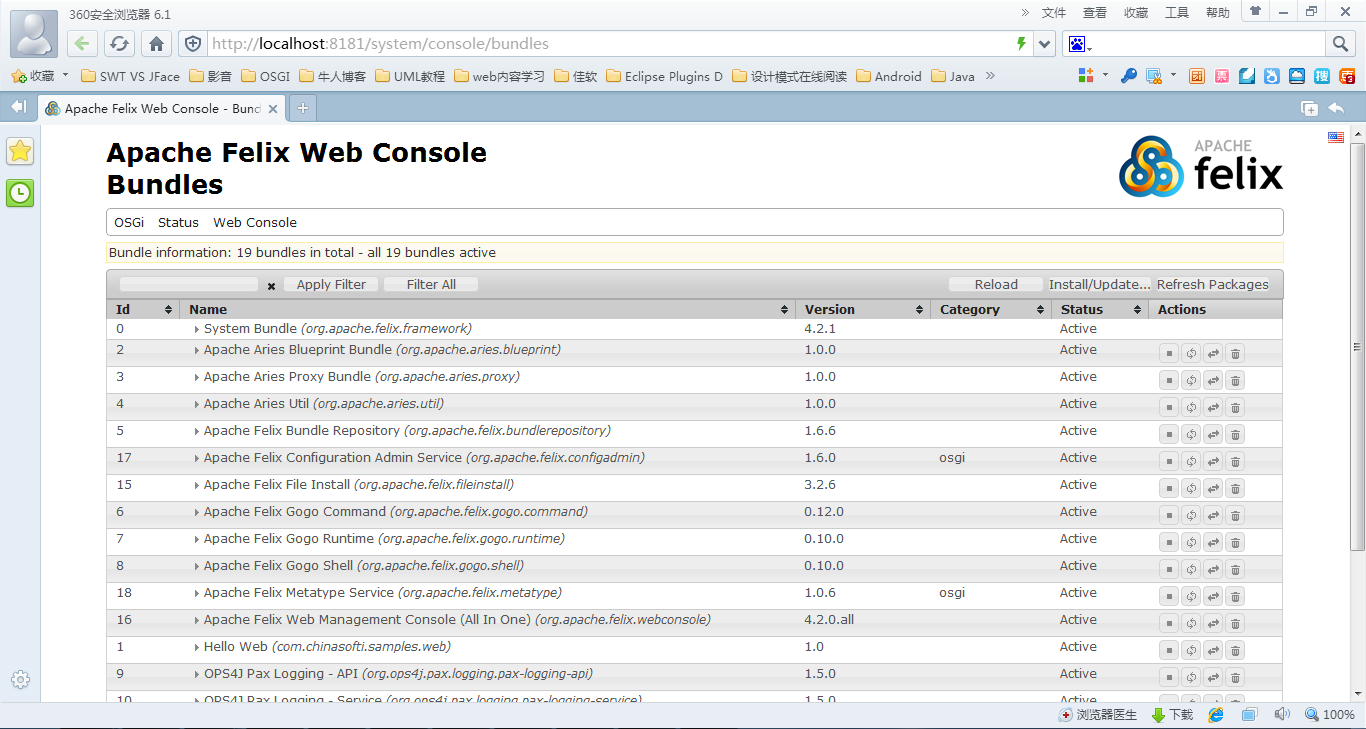
要使用Web Console，需要安装如下所列bundle：

* org.apache.felix.http.jetty-2.2.0.jar
* org.apache.felix.webconsole-4.2.0-all.jar
* org.apache.felix.metatype-1.0.6.jar [可选]
* org.apache.felix.configadmin-1.6.0.jar [可选]

启动felix框架。在浏览器地址栏中键入<http://localhost:8080/system/console>。就可以访问felix的控制台了。



默认登录用户和密码为amdin/admin。登录后界面如下：



更多介绍与使用在如下页面有详述：

<http://felix.apache.org/site/apache-felix-web-console.html>

## Shell

包括如下几个项目，使用起来大同小异。

* [Gogo](http://felix.apache.org/site/apache-felix-gogo.html) - 用来和OSGi交互的较为常用的shell，除了基本的框架命令，其中还包含grep，cat，echo这类指令，并且可以自定义宏命令。
* [Shell](http://felix.apache.org/site/apache-felix-shell.html) - 这个子项目只提供了单纯的shell服务API，而没有提供用户界面。
* [Shell TUI](http://felix.apache.org/site/apache-felix-shell-tui.html) - Apache Felix Shell的文字界面，相比gogo来说更为简单，但是不怎么常用，他的使用依赖于shell bundle。
* [Remote Shell](http://felix.apache.org/site/apache-felix-remote-shell.html) - Apache Felix Shell的远程文字界面，我们通过这个工具可以在其他终端上远程操作Felix。

## File Install

File Install Bundle可以为OSGi框架提供如下功能：可在osgi框架运行过程中，监控某目录bundle的变化，并将其自动安装。从而实现系统真正意义上的自动运行。

其对应的bundle为org.apache.felix.fileinstall-3.2.6.jar，直接将其扔至FELIX\_HOME\bunlde目录下，启动felix框架即可。

可以在FELIX\_HOME\conf\config.properties中对监控目录进行配置。比如

felix.fileinstall.dir=deploy

有关File Install Bundle更多特性配置在如下页面有详述：

<http://felix.apache.org/site/apache-felix-file-install.html>

## Lightweight Http

<http://felix.apache.org/documentation/subprojects/apache-felix-lightweight-http-service.html#apache-felix-lightweight-http-service>

## BluePrint XML配置Gemini Blueprint与Blueprint的区别

| **Element/Attribute** | **Gemini Blueprint** | **Blueprint** |
| --- | --- | --- |
| Namespace Declaration | http://www.springframework.org/schema/beans  or http://www.springframework.org/schema/osgi | http://www.osgi.org/xmlns/blueprint/v1.0.0 |
| Root Element | <beans> | <blueprint> |
| Default Lazy | default-lazy | default-activation |
| Default Init Method | default-init-method | - |
| Default Destroy Method | default-destroy-method | - |
| Default Autowire Strategy | default-autowire, default-autowire-candidates | - |
| Root Element | <beans> | <blueprint> |
| Bean ID | id | id |
| Bean Name/Alias | name/<alias> | - |
| Bean Class | class | class |
| Bean Scope Name | scope | scope |
| Built-in Scopes | singleton, prototype, request, session, bundle | singleton, prototype |
| Lazy Initialization Name/Values | lazy-init=true/false | activation=lazy/eager |
| Depends | depends-on | depends-on |
| Init Method | init-method | init-method |
| Destroy Method | destroy-method | destroy-method |
| Factory Method | factory-method | factory-method |
| Factory Bean | factory-bean | factory-ref |
| Bean Inheritance | parent | - |
| Autowire Strategy | autowire, autowire-candidate | - |
| Constructor | <constructor-arg> | <argument> |
| Property | <property> | <property> |
| Value | <value> | <value> |
| Service Exporter | <service> | <service> |
| Service Importer | <reference>/<list>/<set> | <reference>/<list> |

# OSGi基础

## 系统Bundle

OSGi框架本身也会以一个Bundle的形式向其他Bundle提供资源、Package和服务，比如已经在书中多次出现的Bundle、BundleContext、FrameworkListener等接口，以及后面将会介绍的EventAdmin、PackageAdmin等服务都是由系统Bundle提供的。OSGi规范规定了系统Bundle的Bundle ID固定为0，Bundle的getLocation()方法返回固定字符串“System Bundle”，这些特征使得任何Bundle都可以很方便地从BundleContex.getBundle(0)或BundleContex.getBundle("System Bundle")方法中获取到系统Bundle的对象实例。

在OSGi容器中，系统Bundle可以认为是一定存在的，每一个Bundle都默认依赖这个系统Bundle。下面列出了Equinox框架的系统Bundle的元数据信息。osgi> headers 0  
Bundle headers:  
 Bundle-Activator = org.eclipse.osgi.framework.internal.core.SystemBundleActivator  
 Bundle-Copyright = Copyright (c) 2003, 2004 IBM Corporation and others. All rights reserved. This program and the accompanying materials  are made available under the terms of the Eclipse Public License v1.0 which accompanies this distribution, and is available at http://www.eclipse.org/legal/epl-v10.html  
 Bundle-Description = OSGi System Bundle  
 Bundle-DocUrl = http://www.eclipse.org  
 Bundle-Localization = systembundle  
 Bundle-ManifestVersion = 2  
 Bundle-Name = OSGi System Bundle  
 Bundle-RequiredExecutionEnvironment = J2SE-1.5,OSGi/Minimum-1.2  
 Bundle-SymbolicName = org.eclipse.osgi; singleton:=true  
 Bundle-Vendor = Eclipse.org - Equinox  
 Bundle-Version = 3.8.0.qualifier  
 Eclipse-BundleShape = jar  
 Eclipse-ExtensibleAPI = true  
 Eclipse-SystemBundle = true  
 Export-Package = org.eclipse.osgi.event;version="1.0",  
……//版面关系省略其他Package  
 Export-Service = org.osgi.service.packageadmin.PackageAdmin,org.osgi.service.permissionadmin.PermissionAdmin,org.osgi.service.startlevel.StartLevel,org.eclipse.osgi.service.debug.DebugOptions  
 Main-Class = org.eclipse.core.runtime.adaptor.EclipseStarter  
 Manifest-Version = 1.0

系统Bundle与OSGi框架密不可分，由于它的特殊性，其生命周期变化过程也与普通Bundle有所区别。以下是OSGi规范对系统Bundle生命周期几个过程执行的动作规定。

启动过程：Bundle的start()方法为空操作，因为OSGi框架一启动，系统Bundle就已经启动。

停止过程：Bundle的stop()方法会立即返回并在另外一条线程中关闭OSGi框架。

更新过程：Bundle的update()方法会立即返回并在另外一条线程中重启OSGi框架。

卸载过程：系统Bundle无法卸载，如果执行了Bundle的uninstall()方法，那么框架会抛出一个BundleException异常。

系统Bundle的启动级别固定为0，这个启动级别是无法使用StartLevel接口中的set-BundleStartLevel()进行修改的；如果这样做了，那么OSGi框架将会抛出一个Illegal-ArgumentException异常。