# 目录

[目录 1](#_Toc365383730)

[版本修订 3](#_Toc365383731)

[1 背景 4](#_Toc365383732)

[2 目的 4](#_Toc365383733)

[3 内容 4](#_Toc365383734)

[4 Blueprint 4](#_Toc365383735)

[4.1 轻量级容器 4](#_Toc365383736)

[4.2 低侵入的编程模型 5](#_Toc365383737)

[4.3 何为BluePrint 5](#_Toc365383738)

[4.4 Blueprint规范 6](#_Toc365383739)

[4.5 Blueprint主流实现 7](#_Toc365383740)

[4.6 BluePrint Container 7](#_Toc365383741)

[4.7 BluePrint Bundle 8](#_Toc365383742)

[4.8 Blueprint XML 8](#_Toc365383743)

[4.8.1 Blueprint XML文件片段 9](#_Toc365383744)

[4.8.2 Bean的装配注入 9](#_Toc365383745)

[4.8.3 发布服务 10](#_Toc365383746)

[4.8.4 引用服务 11](#_Toc365383747)

[4.8.5 服务的选择与代理 11](#_Toc365383748)

[4.8.6 服务级别 11](#_Toc365383749)

[5 Apache Aries项目 11](#_Toc365383750)

[6 Apche Aries实践 12](#_Toc365383751)

[6.1 Apache aries blueprint最少需要哪些bundle支持 12](#_Toc365383752)

[6.2 集成JPA 12](#_Toc365383753)

[6.2.1 什么是JPA 12](#_Toc365383754)

[6.2.2 JPA的主流实现 13](#_Toc365383755)

[6.2.2.1 Apache OpenJPA 13](#_Toc365383756)

[6.2.2.2 EclipseLink 13](#_Toc365383757)

[6.2.2.3 Hibernate 13](#_Toc365383758)

[6.2.2.4 Oracle TopLink 13](#_Toc365383759)

[6.2.3 Apache aries集成JPA需要哪些bundle 14](#_Toc365383760)

[6.2.4 使用Apache Aries JPA 15](#_Toc365383761)

[6.2.5 数据源配置 15](#_Toc365383762)

[6.2.6 Persistence Bundle(持久化Bundle) 17](#_Toc365383763)

[6.2.7 注入持久化单元 18](#_Toc365383764)

[6.3 集成JNDI 20](#_Toc365383765)

[6.3.1 什么是JNDI 20](#_Toc365383766)

[6.3.2 Apache Aries集成JNDI 20](#_Toc365383767)

[6.3.2.1 如何配置JNDI服务 21](#_Toc365383768)

[6.3.3 通过JNDI获取Blueprint Container中的服务 21](#_Toc365383769)

[6.3.4 如何获取指定服务 21](#_Toc365383770)

[6.4 集成Transaction 22](#_Toc365383771)

[6.4.1 事务与事务管理 22](#_Toc365383772)

[6.4.2 RESOURCE\_LOCAL和JTA 23](#_Toc365383773)

[6.4.2.1 RESOURCE\_LOCAL 24](#_Toc365383774)

[6.4.2.2 JTA 24](#_Toc365383775)

[6.4.3 JPA中的事务 25](#_Toc365383776)

[6.4.4 Apache aries集成transaction需要哪些bundle 26](#_Toc365383777)

[6.4.5 在JPA中使用事务 26](#_Toc365383778)

[7 结论 26](#_Toc365383779)

[8 总结 27](#_Toc365383780)

# 版本修订

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 描述 | 修订人 |
| V1.0 | 2013-7-1 | 创建 | 陈绪绍 |
| V1.1 | 2013-7-17 | 完善了事务部分内容；增加了结论，总结章节内容 | 陈绪绍 |
| V1.2 | 2013-7-26 | 增加服务的选择与代理，服务级别内容 | 陈绪绍 |
| V1.3 | 2013/8/27 | 添加了  RESOURCE\_LOCAL和JTA内容的描述，在OSGi环境中使用JTA | 陈绪绍 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 背景

基于物流平台产品前期所定的有关OSGi的基本框架选型：Apache felix + Apache Aries。

# 目的

OSGi企业级高级组件Apache Aries在物流平台产品技术层面占有很重要的地位，是贯通其它技术组件的枢纽，如JPA、ESB等。为尽量减小后期开发风险，有必要对Apache Aries的运行环境，使用方法，组件集成等方面进行预研。

# 内容

1. BluePrint XML。
2. Blueprint的基本应用、JPA、JNDI、Transaction等
3. 验证blueprint JPA、JNDI、Transaction应用所需最小运行环境(即bundle集合)。
4. 调研过程中所涉及的其它技术也有涉及，如JPA、OpenJPA等

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。

# Blueprint

## 轻量级容器

2003年左右，轻量级、控制反转（IoC）开始出现，如Spring和Apache Avalon。这一系列IoC容器背后的主要思想是，通过优先使用接口避免使用具体类型的方式简化组件的配置和组装。与IoC容器联合使用的还有依赖注入技术，该技术使得组件依赖于接口类型，接口的实现被注入到组件的实例中。OSGi服务进一步改进了类似的基于接口的方式，但它采用了一种服务定位器（service-locator）模式，并以此来打破组件之间对具体实现的依赖，这与Apache Avalon类似。

与此同时，Service Binder项目为OSGi组件提供了依赖注入框架。由此不难理解为何会出现对二者的比较。无论如何，OSGi对基于接口的服务以及服务定位器模式的使用远超前于当前趋势，没有哪一项技术能够像OSGi那样提供一种完备的动态模块层。现在IoC厂商正经历着重大转变，将他们的基础架构向OSGi框架迁移。

## 低侵入的编程模型

程序员通常需要通过模块化系统API（比如OSGi框架的API）的调用，来实现运行时的有关特性。对API的依赖使得模块非常重量级，脱离了运行时系统，难以测试和执行。为了减少对模块化系统API的依赖，需要提供一个抽象层次，让代码无需直接调用框架API。

事实上，目前也有一些这样的框架和技术，比如[OSGi Blueprint Service](http://elevenframework.org/article/7289212)，Declarative Service， Spring Dynamic Modules和iPojo。这些框架允许你基于POJO来实现模块化的系统，所提供的编程模型使得模块化系统的编程和测试更加方便。

Blueprint允许开发者使用OSGi技术而无需考虑其编程模型。基于Blueprint，我们的代码可以独立于OSGi的API。

## 何为BluePrint

OSGi 4.2 规范定义了Blueprint，该技术其实就是Spring Dynamic Modules的演化和标准化。Blueprint容器（提供依赖注入和反转控制的容器，类似于Spring的功能）规范http://eclipse.org/proposals/gemini/，这一项目将主要实现OSGi企业专家组（OSGi Enterprise Expert Group）制定的各种规范。目前，它主要由SpringSource和Oracle领衔。从目前公开的信息来看，这一开源项目主要专注于OSGi的如下两方面问题。

其一，如何将成熟的Java EE技术集成到OSGi环境中。

其二，实现OSGi 4.2中的各种企业规范，比如OSGi Web容器、 Blueprint Services等；

可以认为，这两个问题是OSGi同Java EE融合的最后两道障碍，一旦消除它们，则企业级Java将迎来新的春天。到那时，不仅是OSGi的春天，也是Java EE的春天。

简单的说，blueprint是一套规范。和Servlet、JSP等J2EE规范一样，是一个约定意义上的东西。最新的OSGi Service Platform Release 4 V4.2 规范包括了一个名为 Blueprint Container 的规范。OSGi企业专家组把 Spring DM 引入的依赖注入 (Dependency Injection) 技术在OSGi服务平台的应用方法进行了定义，为 Bundle 提供一个基于 XML 编写的配置文件，位于路径“OSGI-INF/blueprint/”下。在这个配置文件中，定义了 Bundle 中 Bean 对象的初始化方式以及它们之间的关系，同时，也声明了其需要引用的服务对象和打算发布的服务。拥有这样一个配置文件的 Bundle 可以称之为 Blueprint Bundle，它可以被 Blueprint Container 发现并进行相关的操作。Blueprint Container 规范为OSGi在企业级的发展提供了更大的便利。使用 Blueprint Container 发布OSGi服务有以下好处：

* 以类似 spring 的方式管理对象 (Dependency Injection 方式 )
* 通过配置发布获取服务，发布获取方式更加灵活
* 更好的OSGi平台
* 更先进的IOC（ Inversion of Control）容器功能
* 轻量级、透明，基于POJO的编程模型

## Blueprint规范

Blueprint允许开发者使用OSGi技术而无需考虑其编程模型。基于Blueprint，我们的代码可以独立于OSGi的API。

[**Container Indepedence**](http://elevenframework.org/article/7289355)

这正是blueprint带给我们的：让我们的代码不依赖于OSGi的API。

[**External Configuration**](http://elevenframework.org/article/7289356)

Blueprint支持我们定义模块的配置，因此我们可以外部定义模块的服务，而无需让模块或者暴露为服务的类与OSGi API耦合。

[**Implementation Factory**](http://elevenframework.org/article/7289308)

Blueprint提供了一个OSGi Service的工厂，支持这些服务的运行时自动动态连接，比如你可以像注入一个bean一样注入一个OSGi Service

## Blueprint主流实现

* Apache Aries
* Eclipse Gemini BlueprintGemini前身就是SpringSource捐献的Spring DM 2.0

## BluePrint Container

Blueprint Container规范定义面向服务的组件模型和Spring紧密联合起来，为OSGi定义了一个依赖注入（dependency injection）框架。这个规范的基础来自与Spring Dynamic Modules项目。它的目的是处理OSGi的动态特性，即服务可以在任何时间变得可用和不可用。该规范的另一个意图是处理Java 对象（POJO），这样相同的对象就可以用于OSGi框架的内部和外部。定义并描述应用程序各个组件的 Blueprint XML 文件对Blueprint 编程模型十分重要。规范描述了组件如何被实例化，以及如何相互连接在一起形成一个可以运行的应用程序。一旦扩展器确定某个包是Blueprint包（需要有一些配置文件）后，它将为这个包创建一个Blueprint Container。这个Blueprint Container 负责完成以下操作：

* 解析 Blueprint XML 文件
* 实例化BluePrint Container
* 将组件连接在一起

Blueprint Container 规范定义了四种主要的组件管理器：一个[bean manager](http://www.ibm.com/developerworks/cn/opensource/os-osgiblueprint/#beanman)、一个[service manager](http://www.ibm.com/developerworks/cn/opensource/os-osgiblueprint/#servman)和两个[service reference managers](http://www.ibm.com/developerworks/cn/opensource/os-osgiblueprint/#srms)。每种管理器都负责创建和管理所创建组件的生命周期。管理器提供了一个组件实例。每个管理器都拥有相应的 XML 元素，用于描述管理器属性。管理器可以是顶级管理器，或者内联在其他管理器定义内。管理器还具有一些通用的属性。

**id**

定义管理器的ID。id 属性是可选属性。如果没有指定的话，将自动生成一个唯一 ID 并分配给顶级管理器。内联管理器被认为是匿名的，因此不允许设置 id 属性。管理器 ID 在 Blueprint Container 内对于所有顶级管理器必须是唯一的。管理器使用 ID 彼此引用。例如，在注入期间，管理器将要求被引用的管理器提供对象，该对象将被注入到管理器正在创建的组件中。

**activation**

这个可选属性为管理器定义激活模式。可支持两种激活模式：

* eager，其中管理器在 Blueprint Container 初始化期间激活。
* lazy，其中管理器按需要激活。

默认情况下，启用 eager 激活模式。然而，通过对 blueprint 元素设置 default-activation 属性，可以为 Blueprint XML 文件内的所有管理器修改默认激活模式。当要求管理器提供其第一个组件实例时，管理器将被激活。当 Blueprint Container 被销毁时，管理器将被解除激活。每个管理器都拥有自己的激活和解除激活步骤。

**dependsOn**

这个可选属性指定了一个管理器 ID 列表。所列出的管理器将在其他管理器之前激活。管理器可以具有显式或隐式的依赖关系。dependsOn属性定义了显式的依赖关系。隐式依赖关系在管理器定义中通过对其他管理器的引用定义。

## BluePrint Bundle

Blueprint Container 规范使用了一个扩展器（extender）模式，借助这种模式，扩展器包监视框架中的Bundle的状态，并根据这些Bundle的状态为它们执行操作。Blueprint 扩展器包等待Bundle被激活，然后检查它们是否是 Blueprint Bundle。如果一个Bundle包含一个或多个 Blueprint XML 文件，那么它就被认为是 Blueprint Bundle。这些 XML 文件位于 OSGI-INF/blueprint/ 目录下，或者在Bundle的清单文件META-INF/MANIFEST.MF中通过头部 Bundle-Blueprint显式指定。

## Blueprint XML

Blueprint XML文件的结构以及元素和属性的命名与Spring XML相比没有很大区别。如下所示：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  <service id="WeatherForecastService"  interface="com.ibm.dw.weatherforecast.api.IWeatherForecastService"  activation="lazy" auto-export="disabled" ranking="0">  <bean class="com.ibm.dw.weatherforecast.service.DummyWeatherForecastService"  init-method="init"></bean>  </service>  </blueprint>  ......  <?xml version="1.0" encoding="UTF-8"?>  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  <bean id="WeatherForecastDelegate"  class="com.ibm.dw.weatherforecast.client.WeatherForecastDelegate"  activation="lazy" scope="singleton">  <property name="service">  <reference interface="com.ibm.dw.weatherforecast.api.IWeatherForecastService"  availability="optional" timeout="1000"></reference>  </property>  </bean>  </blueprint> |

Blueprint及Blueprint XML相关内容，可查看osgi企业级规范，可以从osgi联盟官方下载到相应的pdf文档。中文方面的内容可参考IBM如下页面文章：

http://www.ibm.com/developerworks/cn/opensource/os-osgiblueprint/

### Blueprint XML文件片段

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <blueprint xmlns=”http://www.osgi.org/xmlns/blueprint/v1.0.0”>  ...  </blueprint> |

如上即为Blueprint的顶层节点，在顶层节点下，你可以定义bean节点。bean节点可以定义为bean或者bean工厂，从bean结点可以获得bean实例。

### Bean的装配注入

下面是一个简单的 Blueprint Bundle 的配置文件

|  |
| --- |
| <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  <bean id="hw" class="com.abc.HelloWorld">  <argument value=”Rex”/>  <argument value=”Wang”/>  <property name=”country” value=”China”/>  <property name=”city” value=”Shanghai”/>  </bean>  </blueprint> |

其中HelloWorld的类定义如下：

|  |
| --- |
| public class HelloWorld {  private String country;  private String city;  public HelloWorld (String FirstName, String FamilyName){  …  }  public setCountry(String country){  this.country = country;  }  public setCity(String city){  this.city = city;  }  } |

因此，上述配置文件的作用相当于：

|  |
| --- |
| HelloWorldhw = new HelloWorld(“Rex”, “Wang”);  hw.setCountry(“China”);  hw.setCity(“Shanghai”); |

bean 元素用于定义一个 POJO 组件，class 属性用于指定要实例化的 java 类名，argument 和 property 子元素分别用于向构造方法传递参数和属性字段的依赖注入，init-method 和 destroy-method 属性用于指定生命周期回调方法，activation 属性用于指定激活策略是 eager 还是 lazy，scope 属性则用于指定实例的创建是单例模式还是原型模式。

### 发布服务

|  |
| --- |
| <bean id="hw" class="com.abc.HelloWorld">  <argument value=”Rex”/>  <argument value=”Wang”/>  <property name=”country” value=”China”/>  <property name=”city” value=”Shanghai”/>  </bean>  <service ref="hw" interface="com.abc.HelloWorld" /> |

表示将对象“hw”发布为接口为“com.abc.HelloWorld”的服务。

Service 元素用于注册提供服务的组件，注册方式可以是使用 ref 属性引用一个已定义的组件或是直接内联定义一个组件，interface 属性用于指定注册的服务接口，activation 属性用于指定服务的激活策略，ranking 属性用于指定服务级别，具有更高级别的服务会被优先选中，registration-listener子元素则用于定义注册监听器。

### 引用服务

在配置文件中，我们也可以对需要的服务进行引用，并且发布自己的服务，比如：

|  |
| --- |
| <reference id="helloservice" interface="com.abc.HelloWorld" /> |

表示引用一个接口为 “com.abc.HelloWorld”的 Service 对象。reference 元素用于定义服务代理以动态引用服务，interface 属性用于指定查找的服务接口，availability 属性用于指定 Blueprint Container 初始化时服务是否必须存在，timeout 属性用于指定服务代理阻塞调用的超时长度，以毫秒为单位，reference-listener 子元素则用于定义引用监听器。

### 服务的选择与代理

### 服务级别

# Apache Aries项目

Apache Aries 项目实现且扩展了OSGi Alliance EEG 所制定的规范(BluePrint)，提供给编程人员一个易于使用的，轻量级的企业级OSGi应用的编程模型。Aries 项目代码的贡献者来自于 IBM，Progress，RedHat，SAP，Ericsson 等几家公司，它吸收了 Apache Geronimo Blueprint Container 的实现，其应用可以部署到 Apache Geronimo 和 Apache Karaf等运行时环境中。

主页：http://aries.apache.org/

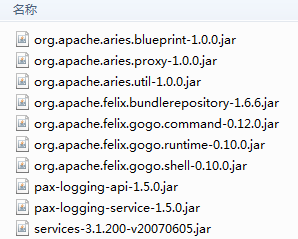
# Apche Aries实践

## 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：



## 集成JPA

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

### 什么是JPA

JPA(JAVA Persistence API)是sun官方提出的Java持久化规范。JPA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。它的出现主要是为了简化现有的持久化开发工作和整合ORM技术，结束现在Hibernate、TopLink、OpenJPA、JDO等ORM框架各自为营的局面。

### 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。

主页：http://openjpa.apache.org/

#### EclipseLink

EclipseLink是由Oracle TopLink捐献给Eclipse社区的开源JPA框架。其官方专门提供了对应的OSGi Bundle。经初步实践，其官方提供的bundle，在equinox环境下，无法被识别。



主页：http://www.eclipse.org/eclipselink/

#### Hibernate

Hibernate 从3.2开始兼容JPA。

主页：http://www.hibernate.org/

#### Oracle TopLink

由Oracle提供的JPA实现框架。TopLink与EclipseLink不同在于，TopLink在EclipseLink基础上增加与Oracle其他产品整合的代码。

主页：http://www.oracle.com/technetwork/middleware/toplink/index.html

### Apache aries集成JPA需要哪些bundle

1. 最基础的Apache Aries相关Bundle，参考6.1
2. Aries相关JPA Bundle

* org.apache.aries.jpa.api-1.0.0.jar
* org.apache.aries.jpa.container.context-1.0.0.jar
* org.apache.aries.jpa.blueprint.aries-1.0.0.jar
* org.apache.aries.jpa.container-1.0.0.jar

1. Aries相关JNDI Bundle

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

1. Aries相关TransactionBundle

* org.apache.aries.transaction.blueprint-1.0.0.jar
* org.apache.aries.transaction.manager-1.0.0.jar
* org.apache.aries.transaction.wrappers-1.0.0.jar

1. 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

* asm-all-3.2.jar
* commons-collections-3.2.1.jar
* commons-lang-2.5.jar
* commons-pool-1.5.4.jar
* openjpa-2.2.2.jar
* javax.persistence\_2.0.5.v201212031355.jar
* commons-dbcp-1.4.jar
* javax.transaction\_1.1.0.jar
* org.apache.servicemix.bundles.serp-1.13.1\_2.jar

1. EclipseLink

有待验证。

1. Hibernate

有待验证。

**说明**：

Apache Aries相关的Bundle可在其官方网站如下页面下载：<http://aries.apache.org/downloads/currentrelease.html>。也可以check Aries源码，利用Maven进行打包构建。

OpenJPA可以去官方网站<http://openjpa.apache.org/>去下载

### 使用Apache Aries JPA

Apache Aries中使用JPA一般通过如下步骤来进行：

1. 设置数据源

参考6.2.5章节。

1. 创建持久化Bundle

参考6.2.6章节

1. 注入持久化单元

参考6.2.7章节。

### 数据源配置

数据源是指对象或工厂，用于生成与应用程序内的实际数据源（通常是数据库）的连接。在 Java Platform, Enterprise Edition (Java EE) 中，此数据源是指数据库连接池，一组随时可用的数据库连接。创建数据源或数据库池的原因是它能够提高应用程序性能，因为即使打开一个连接也会十分占用CPU。

通常我们在开发基于 OSGI 框架下的 JPA 应用程序时，会创建一个单独的 bundle ，将数据源包装成 OSGI 的 service，提供给其它 bundle，并通过JNDI进行注册。

下面列出主流数据库在BluePrint环境下数据源的配置。需要提供相应数据库的jdbc驱动包。

##### 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

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  <bean id="dataSource" class="oracle.jdbc.pool.OracleDataSource">  <property name="URL" value="jdbc:oracle:thin:@localhost:1521:orcl"/>  <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/oracleds"/>  </service-properties>  </service>  </blueprint> |

##### DB2

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">  <bean id="dataSource" class="com.ibm.db2.jcc.DB2DataSource">  <property name="url" value="jdbc:db2://localhost:5021/sample"/>  <property name="user" value="db2admin"/>  <property name="password" value="db2admin"/>  </bean>  <service interface="javax.sql.DataSource" ref="dataSource">  <service-properties>  <entry key="osgi.jndi.service.name" value="jdbc/db2ds"/>  </service-properties>  </service>  </blueprint> |

### Persistence Bundle(持久化Bundle)

Apache Aries JPA Container为OSGi应用提供了JPA支持。一个持久性Bundle定义了JPA 运行时会使用的持久化类和持久化单元。持久化bundle首先是一个OSGi Bundle。

它包含一个或多个persistence.xml 文件，并且在的 META-INF/MANIFEST.MF文件中，会包含一个 Meta-Persistence头。持久化配置文件 persistence.xml 可位于OSGi Bundle中的任何位置。默认位置是 META-INF/persistence.xml；此时Meta-Persistence内容为空(有一个空格)。Meta-Persistence头要列举出 persistence bundle 中 persistence.xml 文件的所有位置，可以使用逗号分隔的路径指定其他位置，例如，Meta-Persistence：jpa.xml, persistence.jar!/JPA/persistence.xml。

如下所示为简单的持久化Bundle的MANIFEST.Mf文件内容：

|  |
| --- |
| Manifest-Version: 1.0  Bundle-ManifestVersion: 2  Bundle-Name: blog.persistence  Bundle-SymbolicName: com.ibm.ws.eba.example.blog.persistence  Bundle-Version: 1.0.0  Meta-Persistence:  Import-Package:  com.ibm.ws.eba.example.blog.persistence.api;version="[1.0.0,1.1.0)",  javax.persistence;version=1.0.0 |

通常我们在开发基于 OSGI 框架下的 JPA 应用程序时，会为持久层创建一个单独的 bundle。将操作 Entity 的 DAO(Database Access Object) 包装成 OSGI 的 service，提供给其它 bundle。

persistence.xml文件定义了持久化单元，如下为位于META-INF目录下的persistence.xml文件内容：

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <persistenceversion=*"1.0"*  xmlns=*"http://java.sun.com/xml/ns/persistence"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd"*>  <!—定义持久化单元🡪  <persistence-unit name=*"default"* transaction-type=*"JTA"*>  　<!—JPA提供商🡪  <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>  <!—通过JNDI引用数据源🡪  <jta-data-source>osgi:service/jdbc/mysqlds</jta-data-source>  <!—持久化实体类🡪  <class>com.chinasofti.samples.biz.User</class>  <exclude-unlisted-classes>true</exclude-unlisted-classes>  　　　　<!—各JPA实现的一定特定参数🡪  <properties>  <propertyname=*"openjpa.jdbc.SynchronizeMappings"*value=*"buildSchema(ForeignKeys=true)"*/>  <propertyname=*"openjpa.RuntimeUnenhancedClasses"*value=*"supported"*/>  <propertyname=*"openjpa.Multithreaded"*value=*"true"*/>  <propertyname=*"openjpa.Log"*  value=*"DefaultLevel=TRACE, Runtime=TRACE, Tool=TRACE, SQL=TRACE"*/>  </properties>  </persistence-unit>  </persistence> |

### 注入持久化单元

Blueprint提供了两个标签对JPA持久化单元注入托管提供了支持。分别是<jpa:unit>和<jpa:context>。其中<jpa:unit>用来注入EntityManagerFactory。<jpa:context>用来注入EntityManager。如下所示即为Blueprint容器托管的 Transaction 和 Persistence 配置

* 注入EntityManager

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <blueprintxmlns=*"http://www.osgi.org/xmlns/blueprint/v1.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns:tx=*"http://aries.apache.org/xmlns/transactions/v1.0.0"*  xmlns:jpa=*"http://aries.apache.org/xmlns/jpa/v1.0.0"*>  <beanid=*"userManager"*class=*"com.chinasofti.samples.jpa.UserManagerImpl"*>  <tx:transaction method=*"\*"*value=*"Required"*/>  <jpa:context property=*"em"* unitname=*"default"*></jpa:context>  </bean>  <serviceref=*"userManager"*interface=*"com.chinasofti.samples.biz.UserManager"*/>  </blueprint> |

对应持久化实现类实现为:

|  |
| --- |
| package com.chinasofti.samples.jpa;  import javax.persistence.EntityManager;  import javax.persistence.PersistenceContext;  import com.chinasofti.samples.biz.User;  import com.chinasofti.samples.biz.UserManager;  public class UserManagerImpl implements UserManager {  private EntityManager em;  public void setEm(EntityManagerem) {  this.em = em;  }  @Override  public void addUser(User user) {  em.persist(user);  }  @Override  public void delUser(User user) {  User tuser = em.find(User.class, user.getId());  em.remove(tuser);  }  } |

其中UserManagerImpl类的变量em被如下标签给注入进来了

<jpa:contextproperty=*"em"*unitname=*"default"*></jpa:context>

* 注入EntityManagerFactory

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <blueprintxmlns=*"http://www.osgi.org/xmlns/blueprint/v1.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns:tx=*"http://aries.apache.org/xmlns/transactions/v1.0.0"*  xmlns:jpa=*"http://aries.apache.org/xmlns/jpa/v1.0.0"*>    <bean id=*"personManager"* class=*"com.chinasofti.samples.jpa.PersonManagerImpl"*>  <tx:transaction method=*"\*"* value=*"Required"*/>  <jpa:junit property=*"em"*unitname=*"default"*/>  </bean>    <service  ref=*"personManager"* interface=*"com.chainsofti.samples.module.entity.PersonManager"*/>  </blueprint> |

对应持久化实现类实现为:

|  |
| --- |
| package com.chinasofti.samples.jpa;  import javax.persistence.EntityManager;  import com.chainsofti.samples.module.entity.Person;  import com.chainsofti.samples.module.entity.PersonManager;  public class PersonManagerImpl implements PersonManager{  private EntityManagerFactoryem;  public void setEm(EntityManagerFactoryem) {  this.em = em;  }  @Override  public void addPerson(Person person) {  EntityManager manager = em.createEntityManager();  manager.persist(person);  }  } |

## 集成JNDI

### 什么是JNDI

Java命名和目录接口（the Java naming and directory interface，JNDI）是一组在Java应用中访问命名和目录服务的API。为开发人员提供了查找和访问各种命名和目录服务的通用、统一的接口。命名服务将名称和对象联系起来，使得开发人员可以用名称访问对象。

JNDI包含一组API和一组SPI(Service Provider Interface)，如基础包javax.naming。Java程序通过JNDI API 存取各种naming和directory服务。

### 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

#### 如何配置JNDI服务

|  |
| --- |
| <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

|  |
| --- |
| InitialContextic = **new**InitialContext();  BloggingService blog= ic.lookup("osgi:service/"+ 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

|  |
| --- |
| InitialContextic = 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。

### 如何获取指定服务

过滤的方式：

InitialContext icx = **null**;

IService service = **null**;

**try** {

icx = **new** InitialContext();

service = (IService) icx.lookup("osgi:service/"

+ IService.class.getName()+"/(name=putinstorage)");

## 集成Transaction

### 事务与事务管理

事务 (Transaction) 是访问并可能更新数据库中各种数据项的一个程序执行单元 (unit)。在关系数据库中，一个事务可以是一条或一组 SQL 语句，甚至整个程序。它有通常被称为 ACID 的原子性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持续性（Durability）四大特性：

原子性（Atomicity）：一个事务是一个不可分割的工作单位，事务中包括的诸操作要么都做，要么都不做。

一致性（Consistency）：事务必须是使数据库从一个一致性状态变到另一个一致性状态。一致性与原子性是密切相关的。

隔离性（Isolation）：一个事务的执行不能被其他事务干扰。即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。

持久性（Durability）：持续性也称永久性（permanence），指一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。接下来的其他操作或故障不应该对其有任何影响。

在 Java EE 的应用开发中，事务的应用是必不可少的，同时由于方法调用的原因，比如方法 A 调用方法 B 的时候。如果方法 A 在事务环境中运行，那么方法 B 是否也要在事务中运行呢，方法 B 是要和方法 A 在同一个事务中运行还是新开起一个事物呢？等等。要弄清楚这些问题，就要牵涉到事务传播属性的问题，EJB 中针对不同的的情况提供了下面六种不同的事物传播属性：

**Required**：用该属性标注的方法或组件总是在事务中运行。如果客户端已经在事务中，则在原事务中运行；如果没有事务，则开启一个新事务，在其中运行。

**Requires\_New**：方法或组件总是在新开启的事务中运行。如果客户端已经在事务中，则首先将原事务挂起，然后新开启一个事务，在其中运行，新事务结束之后，原来事务从挂起点继续执行；如果没有事务，则开启一个新事务，在其中运行。

**Supports**：和 Required 属性的不同点是，在没有事务的环境中不会开启一个新事务；如果存在事务的话则加入其中运行，这点和 Reuqired 相同。

**Not\_Supported**：如果事务已经存在的情况下，则原来的事务要挂起，然后调用标注该属性的方法或组件，调用结束之后，继续原来的事务；无事务环境中调用的时候，不开启新事务，这点和 Supports 相同。

**Mandatory**：调用标注该属性的方法或组件的客户端，必须已经在事务中，如果不在事务中则会抛出异常；如果已经在事务中，则加入原来事务运行。和 Required 不同的是，该属性不会自动开启新的事务；

Never**：**用 Never 属性标注的方法或组件，不能在事务中运行。如果调用该方法或组件的客户端已经在事务中，则抛出异常。

事务管理是对一系列操作的管理，它最终只有两个结果，要么成功，要么失败。一旦失败，所有的操作将回滚到初始状态。一旦成功，才最终提交，最终持久化。事务管理对银行系统最为典型。例如一个人去银行取款，他取款的钱此时大于银行账户中的钱，此时交易失败，所以取款不成功，事务回滚到操作前的状态。

在JPA中，对于实体的“CRUD”基本操作，其中涉及事务的是“C”、“U”和“D”，即“新建”、“更新”和“删除”，因为这些操作都会影响数据库中的数据变化，所以必须使用事务保证其一致性；对于“R”查询，只是查询数据，没有对数据产生变化，所以并不需要控制事务。

### RESOURCE\_LOCAL和JTA

#### RESOURCE\_LOCAL

RESOURCE\_LOCAL事务数据库本地的事务。它是数据库级别的事务，只能针对一种数据库，不支持分布式的事务。对于中小型的应用，可以采用RESOURCE\_LOCAL管理EntityManager事务。通常在J2SE环境中都是使用RESOURCE\_LOCAL事务。

在J2SE环境中，使用RESOURCE\_LOCAL管理EntityManager事务的代码如下所示：

|  |
| --- |
| public class JPAClient {      public static void main(String[] args) {          EntityManagerFactory emf = Persistence.createEntityManagerFactory("default");          EntityManager entityManager = emf.createEntityManager();          try {              /\*\* 事务开始 \*/              entityManager.getTransaction().begin();              CustomerEO customer = new CustomerEO();              customer.setName("Janet");              customer.setEmail("");              customer.setAsset(100000.00);              /\*\* 事务提交 \*/              entityManager.getTransaction().commit();          } finally {              entityManager.close();              emf.close();          }      }  } |

#### JTA

* + 什么是 Java Transaction API（JTA）？

JTA 是事务服务的 J2EE 解决方案。本质上，它是描述事务接口（比如 UserTransaction 接口，开发人员直接使用该接口或者通过 J2EE 容器使用该接口来确保业务逻辑能够可靠地运行）的 J2EE 模型的一部分。

JTA 具有的三个主要的接口分别是 UserTransaction 接口、 TransactionManager 接口和 Transaction 接口。这些接口共享公共的事务操作，例如 commit() 和 rollback() ， 但是也包含特殊的事务操作，例如 suspend() ， resume() 和 enlist() ，它们只出现在特定的接口上，以便在实现中允许一定程度的访问控制。例如， UserTransaction 能够执行事务划分和基本的事务操作，而TransactionManager 能够执行上下文管理。它是容器级别的事务。它的最大优势是可以支持分布式的事务，如果系统采用的是分布式的数据库，那么只能选择JTA管理EntityManager事务。

* + **在J2EE和OSGi环境下使用JTA**

JTA 是一个在 Java EE 和 OSGi 中使用的标准事务 API，支持在一个 Java 环境下跨多个 XA 资源执行分布式事务。基于 JTA 的事务可以由应用程序（使用 javax.transaction.UserTransaction API）管理，或由应用程序服务器管理、也可以由事务管理器（使用 javax.transaction.TransactionManager API）管理。在这里，事务现在由 TransactionManager 管理。

### JPA中的事务

EntityManager对象的事务管理方式有两种，分别为JTA和RESOURCE\_LOCAL，即Java Transaction API方法和本地的事务管理。

JPA中的事务类型通过persistence.xml文件中的“transaction-type”元素配置。如果 transaction-type 为 JTA，则实体管理器可以通过调用 joinTransaction() 方法加入事务。RESOURCE\_LOCAL类型通常在非J2SE环境中使用，JTA通用在J2EE和OSGi环境中使用。

例如，配置事务为JTA方式的代码如下所示。

|  |
| --- |
| <persistence>      <persistence-unit name="demo" transaction-type="JTA">          //其他配置省略      </persistence-unit>  </persistence> |

如果使用RESOURCE\_LOCAL管理事务，则配置代码如下所示。

|  |
| --- |
| <persistence> <persistence-unit name="demo" transaction-type="RESOURCE\_LOCAL">  //其他配置省略 </persistence-unit> </persistence> |

### Apache aries集成transaction需要哪些bundle

* org.apache.aries.transaction.blueprint-1.0.0.jar
* org.apache.aries.transaction.manager-1.0.0.jar
* org.apache.aries.transaction.wrappers-1.0.0.jar

### 在JPA中使用事务

为了保证数据库操作相关的类中所有的方法都能在 blueprint 容器中所建立的全局事务中运行，还需要做以下的配置：

* 在 blueprint 元素中添加命名空间属性：

xmlns:tx="http://aries.apache.org/xmlns/transactions/v1.0.0"

* 同时添加以下元素作为<bean …></bean>子元素：

<tx:transaction method="\*" value="Required"/>

value可使用的值为6.4.1中描述的6种，但一般情况下都是使用的Required。

如下即为集成JTA的示例配置文件内容：

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <blueprintxmlns=*"http://www.osgi.org/xmlns/blueprint/v1.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*xmlns:tx=*"http://aries.apache.org/xmlns/transactions/v1.0.0"*  xmlns:jpa=*"http://aries.apache.org/xmlns/jpa/v1.0.0"*>  <beanid=*"userManager"*class=*"com.chinasofti.samples.jpa.UserManagerImpl"*>  <tx:transaction method=*"\*"*value=*"Required"*/>  <!-- <tx:transaction method="persistName" value="Required"/> -->  <jpa:contextproperty=*"em"*unitname=*"default"*></jpa:context>  </bean>  <serviceref=*"userManager"*interface=*"com.chinasofti.samples.biz.UserManager"*/>  </blueprint> |

# 结论

利用Apache Aries提供一系列技术，在OSGi中使用JPA，JNDI、Transaction上手曲线是较为平缓的，只需要熟悉一些标签的使用，就能够很方便的对JPA进行托管和注入，瓶颈在于是否能够熟练使用JPA以及对相应的JPA实现是否有足够的理解，在OSGi中使用JPA，对JPA较为熟悉是最基本的要求。最好是使用过ORM框架。

# 总结

本文档所涉及的最核心的内容是JPA的集成，JNDI和Transaction的集成和使用较为简单，是和JPA搭配使用的。另外blueprint xml的配置也是一块较为重要的内容。限于时间，对blueprint xml的介绍只描述了部分内容。

当前只验证了基于equinox内核+openjpa+apachearies的组合。在felix中，抛出包依赖错误，但经过查看相关bundle依赖版本，暂没有发现提示所报错误。在equinox下没有此问题。限于时间，初步选择了在equinox下跑，apache felix下需要后续再做验证。

JPA框架的验证尝试了OpenJPA、Hibernate、EclipseLink，最终验证成功的只有OpenJPA。Hibernate、EclipseLink在验证过程中，或多或少存在一些问题，比如依赖包版本冲突，bundle无法被识别等。限于时间，没有继续去验证。

　在预研的过程中，遭遇OPenJPA需要对所有实体类在编译时进行enhance（增强字节码）的问题。后面经过查阅OpenJPA官方文档及多次实践，通过调整OSGi实现版本，问题暂时得以解决(相关细节，在如下页面也有提及：http://aries.apache.org/modules/jpaproject.html)。另外，OpenJPA官方文档上提及，增强实体类能够提高数据库操作效率，这一点也需要在后面进行了解。