密 级：内部公开

文档编号：

Jetspeed模块分析文档

|  |  |
| --- | --- |
| 编制： | 生效日期： |
| 审核： | 批准： |

深圳市金蝶中间件有限公司对本文件资料享受著作权及其它专属权利，未经书面许可，不得将该等文件资料（其全部或任何部分）披露予任何第三方，或进行修改后使用。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件更改摘要： | | | | | |
| 日期 | 版本号 | 修订说明 | 修订人 | 审核人 | 批准人 |
| 2013-1-1 | 1.0 | APS后台设计文档第一版 | 袁星勤 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 目录

[第1章 简介 4](#_Toc341795813)

[1.1 目的 4](#_Toc341795814)

[1.2 适用对象 4](#_Toc341795815)

[1.3 参考资料 4](#_Toc341795816)

[第2章 整体描述 5](#_Toc341795817)

[2.1 系统结构 5](#_Toc341795818)

[第3章 模块分析 6](#_Toc341795819)

[3.1 jetspeed-cm 6](#_Toc341795820)

[3.1.1 基本设计概述 6](#_Toc341795821)

[3.1.2 设计思想 6](#_Toc341795822)

[3.1.3 系统处理流程 6](#_Toc341795823)

[3.1.4 重要数据模型 6](#_Toc341795824)

[3.1.5 重要业务流程实现 6](#_Toc341795825)

[3.1.6 参考资料 6](#_Toc341795826)

[3.2 jetspeed-deploy-tools 7](#_Toc341795827)

[3.2.1 基本设计概述 7](#_Toc341795828)

[3.2.2 设计思想 7](#_Toc341795829)

[3.2.3 系统处理流程 7](#_Toc341795830)

[3.2.4 重要数据模型 7](#_Toc341795831)

[3.2.5 重要业务流程实现 7](#_Toc341795832)

[3.2.6 参考资料 7](#_Toc341795833)

# 简介

## 简介

本文档对jetspeed2.2.2版本进行深入研究，以便更好地利用jetspeed已有功能和改造jetspeed来满足APS的需求。

## 适用对象

本文档适用于：系统设计师，开发工程师。

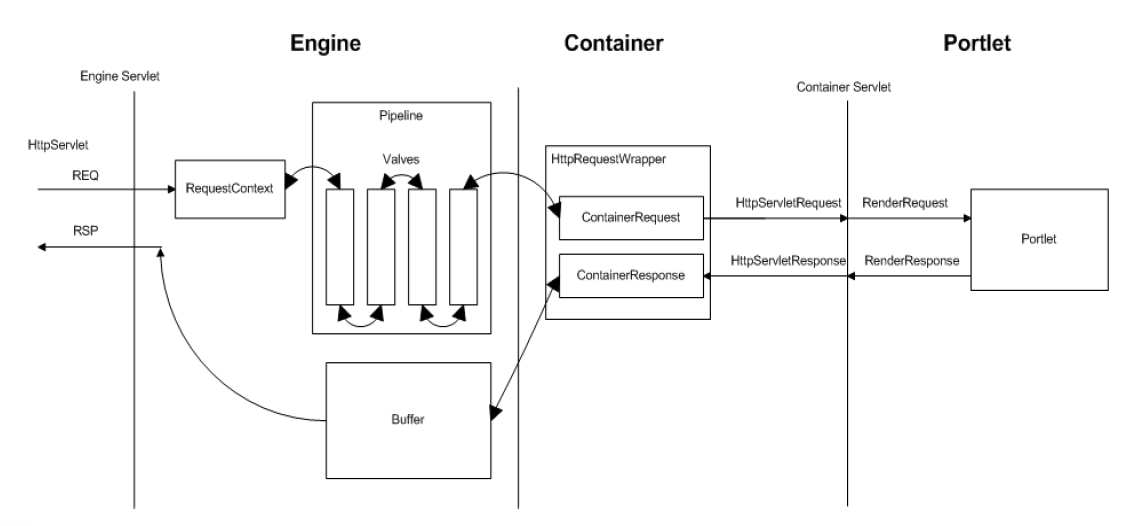
## 参考资料

表结构/APS\_Version\_2.0.0.sql

# 整体描述

## 从请求机制看Jetspeed

首先看一下Jetspeed的架构图。



在这个图中，Jetspeed Portal对应 Engine这部分，Pluto Portlet Container对应Container这一部分，而Portlet则代表Portlet小程序。这张图正好对应了JSR168/286中描述的Portal 🡪 Portlet Container 🡪 Portlet 的关系。

从上图可以看到，通常我们研究jetspeed指的是Engine部分，更具体一点就是pipeline。

结合Jetspeed中的代码，对Jetspeed的架构图中的处理流程可以解读如下：

1客户端发送的request送至Portal server后由JetspeedServlet(org.apache.jetspeed.JetspeedServlet)接收

2 JetspeedServlet通过Jetspeed(org.apache.jetspeed.Jetspeed)取得 ComponentManager，然后通过ComponentManager取得 RequestContextComponent(org.apache.jetspeed.request.JetspeedRequestContextComponent)

3 RequestContextComponent会针对这个 request建立一个RequestContext(org.apache.jetspeed.JetspeedRequestContext)，并且让这个request和request context互相参照。

4调用Engine(org.apache.jetspeed.JetspeedEngine)的service()方法，上面产生的RequestContext会被传入作为参数。在service()方法中，Engine会根据目标URL通过Components Manager来获取相应的Pipeline(org.apache.jetspeed.pipelineJetspeedPipeline)处理。Pipeline（管道）使用了 chain of responsibility的设计模式，每一个Pipeline是由一串Valve(org.apache.jetspeed.pipeline.valve.Valve，阀门)组成。

5 Pipeline中的各个Valve依次执行，各自完成自己的功能。其中的一些Valve会和Pluto Container交互，某些Valve负责产生portal页面等等。在这个过程中，Container会执行相关的Portlet，并将Portlet返回的Fragment(片段)放到一个Buffer中。

1. Portal server负责将返回的片段进行组装成页面，返回给客户端。

## Pipeline

**Pipeline**的中文意思是“管道”。**Pipeline**是引擎的核心。

Jetspeed使用了管道机制对一个请求进行处理。Pipeline背后的思想是”Chain of Responsibility”的设计模式。Jetspeed处理一个请求的过程就好像是一辆汽车产品在流水线上组装的过程。在流水线作业中，根据设计蓝图，第一个工人负责搭建骨架，第二个工人负责安装轮子，第三个工人安装发动机。。。。流水线上的每个工人都在上一个人的工作基础上进行加工，当最后一个工人完成工作时，一辆汽车也就组装完成了。在Jetspeed的处理机制中，汽车蓝图就是RequestContext, 流水线就是Pipeline（管道），而流水线上的工人就是Valve(阀门)。每一个汽车产品的组装都有不同的流水线，在Jetspeed中处理每一个RequestContext都拥有不同的Pipeline实例。

Jetspeed定义了不同的Pipeline类型。而每一种pipeline都有不同的valves组合。关于Jetspeed如何决定使用哪个pipeline，具体参考：org.apache.jetspeed.engine.JetspeedEngine的service()方法。

Pipelines定义文件：assembly\pipelines.xml，该文件定义了Pipeline和valve，以及Pipeline和servelt path的映射关系。

提示：以上我们知道Jetspeed的各种功能都是通过valve完成的。要跟踪Jetspeed的某些具体功能的实现细节，都可以从对应的valve的invoke()方法开始。

官方资料：<http://portals.apache.org/jetspeed-2/guides/guide-components.html>

## Jetspeed的配置文件

Jetspeed引擎的配置文件都在jetspeed-portal-resources工程下。

## Jetspeed的启动过程

org.apache.jetspeed.engine.JetspeedServlet是Jetspeed的入口。当启动Jetspeed应用的时候，由这个Servlet负责初始化Jetspeed运行所需额环境和资源。启动的大概步骤如下：

1. 装载“WEB-INF/conf/ 下的三个配置文件：jetspeed.properties,override.properties, spring-filter-key.properties

//To do: 这个三个文件分别配置了什么内容？

1. 初始化一个ComponentManager。Jetspeed使用了Spring作为自身组件的管理引擎。你也可以换成其它组件管理引擎。具体的做法是：覆盖JetspeedServlet类的protected ComponentManager initializeComponentManager(ServletConfig servletConfig, String appRoot, Configuration configuration) throws IOException

方法。

初始化主要根据“WEB-INF/assembly/\*.xml, /WEB-INF/assembly/override/\*.xml的配置，生成一个org.apache.jetspeed.components.SpringComponentManage的实例。

1. 生成一个JetspeedEngine实例，并启动它。请参考“JetspeedEngine的start()方法。

JetspeedEngine在启动的时候做了如下事情：

1. 由初始化ComponentManager，由ComponentManager把“WEB-INF/assembly/\*.xml, /WEB-INF/assembly/override/\*.xml”中配置的组件注册到运行环境中。

在JetspeedEngine和ComponentManager被初始化之后，Jetspeed本身的初始化已经完成。接下就是装载portlet应用。在Jetspeed中，负责装载portlet应用的类是：org.apache.jetspeed.container.JetspeedContainerServlet

观察每一个Portlet的应用，你可以在他们的 WEB-INF/web.xml下发现以下配置：

<servlet>

<description>MVC Servlet for Jetspeed Portlet Applications</description>

<display-name>Jetspeed Container</display-name>

<servlet-name>JetspeedContainer</servlet-name>

<servlet-class>org.apache.jetspeed.container.JetspeedContainerServlet</servlet-class>

<init-param>

<param-name>contextName</param-name>

<param-value>dbbrowser</param-value>

</init-param>

<load-on-startup>0</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>JetspeedContainer</servlet-name>

<url-pattern>/container/\*</url-pattern>

</servlet-mapping>

如果一个web应用（可以包含很多portlet）要成为一个可以被Jetspeed识别的Portlet, 以上配置是必不可少的。事实上，当通过Jetspeed发布一个Portlet时（把war包放在jetspeed\WEB-INF\deploy下），jetspeed会自动在web.xml添加以上信息（如果还没有被配置）。

最终一个Portlet应用的启动时通过org.apache.jetspeed.tools.pamanager.PortletApplicationManager的StartApplication(…)方法来启动的。这个方法利用了spring的MethodInvokingFactoryBean来启动。我们可以在jetspeed\WEB-INF\assembly\deployment.xml中找到如下配置：

<!-- Inject PAM Proxy into PAM implementation -->

<bean id="injectPAMProxyIntoPAM" class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">

<meta key="j2:cat" value="default" />

<property name="targetObject">

<ref bean="org.apache.jetspeed.tools.pamanager.PortletApplicationManager" />

</property>

<property name="targetMethod">

<value>setPAMProxy</value>

</property>

<property name="arguments">

<list>

<ref bean="PAM" />

</list>

</property>

</bean>

在每一个portlet应用启动完成之后，JetspeedServlet开始监听请求。请看JetspeedServlet的doGet()方法。

我们可以发现以下事实：

1. Portal的context path是在第一次请求发生时才初始化的。

针对每一个请求，Jetspeed都会通过ComponentManager生成一个org.apache.jetspeed.request.JetspeedRequestContext 对象。从它的javadoc我们可以看到它的作用：

/\*\*

\*JetspeedRequestContextisassociatedwitheachportalrequest.Therequest

\*holdsthecontextualinformationsharedamongstcomponentsintheportal,

\*accessedthroughacommonvalvepipeline.

\*

\*/

在获得一个JetspeedRequestContext的对象后，然后调用Jetspeed的service方法来处理这个请求。

核心代码：

try

{

context = contextComponent.create(req, res, getServletConfig());

engine.service(context);

}

finally

{

contextComponent.setRequestContext(null);

}

Service()方法使用ComponentManager根据目标URL生成相应的pipeline, 然后由Pipeline中的Valve对这个请求进行处理。

## Portlet Bridges

所谓portlet桥，就是结合某些框架或技术来开发portlet。

Struts Portlet Bridge

JSF Portlet Bridge

Velocity Portlet Bridge

Perl Portlet Bridge

PHP Portlet Bridge

官方资料：<http://portals.apache.org/jetspeed-2/devguide/guide-portlet-bridges.html>

**后继工作:**

**Jetspeed的渲染机制（layout， psml）**

**窗体状态变化时，处理流程怎么样？**

**在portlet里面提交form时，如何工作？**

**如何把portlet返回的片段嵌入psml中**

**Psml的整套机制**

**Jetspeed的事件处理机制**

**Jetspeed的安全机制**

**Jetspeed的desktop版本是如何工作的？**

# 模块分析

## jetspeed-cm

### 设计概述

1、模块依赖

该模块被jetspeed-rdbms和jetspeed-serializer模块所依赖。

2、整体功能描述

a）在Jetspeed启动时，启动spring容器，把\WEB-INF\assembly下的所有bean文件加载进来，并且通过spring-filter.properties指定的分类过滤，将属于这些分类的bean进行初始化

b）Spring调用ehcache对portlet内容、装饰等bean进行缓存配置

### 系统处理流程

本条描述模块的主要处理流程，

1、spring容器初始化流程

Jetspeed使用了Spring作为自身组件的管理引擎。Jetspeed的initializeComponentManager()初始化了spring容器。

初始化主要根据“WEB-INF/assembly/\*.xml, /WEB-INF/assembly/override/\*.xml的配置，生成一个org.apache.jetspeed.components.SpringComponentManager的实例。

由ComponentManager把“WEB-INF/assembly/\*.xml, /WEB-INF/assembly/override/\*.xml”中配置的组件注册到运行环境中。

流程图



### 设计思想

1、深入分析

2、关键技术、核心算法、思想

3、类图

SpringComponentManager：负责初始化spring容器，加载过滤后的bean。

EhCacheImpl：jetspeed通过ehcache包装了自己的缓存

EhCacheDistributedImpl：jetspeed通过ehcache包装了自己的分布式缓存

JetspeedCacheMonitorImpl：监控所有cache，并且可以对这些cache进行统计，计算和重置



### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表

（无）

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-deploy-tools

### 设计概述

1、模块依赖

jetspeed-deploy-tools是一个独立的模块，该模块被jetspeed-portal模块所依赖。

2、整体功能描述

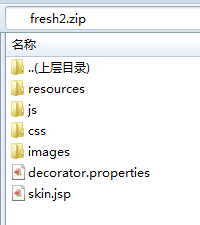
部署可以简单的分为三个部分，首先是把应用copy到jetspeed的指定目录，由jetspeed再copy到容器可以访问的目录，并由容器启动应用。然后，如果是portlet应用，则调用jetspeed的portlet注册类，把应用注册到jetspeed中。最后就是提供一个管理接口，可以由用户控制应用的停止、启动、重启、卸载等。

名称解析：jetspeed支持两种类型的portlet应用，姑且称之为local portlet和servlet portlet。至于servlet portlet应用和local portlet应用的区别，设计思想章节会详细介绍。

jetspeed-portal模块会对引擎下WEB-INF\deploy和WEB-INF\deploy\local两个目录进行循环扫描，然后调用jetspeed-deploy-tools模块的核心类JetspeedDeploy对扫描到的portlet应用或decorator jar(zip)文件进行部署。

所谓的部署，其实是读取web.xml，portlet.xml，context.xml，然后进行简单的校验和节点修正，最后把这些配置文件和其他文件一起写到部署目录下，对于servlet portlet应用，需要servlet容器启动该应用，然后注册到jetspeed。Local portlet则直接注册到jetspeed。

1. WEB-INF\deploy和WEB-INF\deploy\local目录。
2. 上传到WEB-INF\deploy的应用被当做是一个servlet portlet应用。
3. 上传到WEB-INF\deploy\local的应用都被当做是local portlet 应用。
4. decorator jar(zip)文件上传到两个目录都可以。
5. Decorator 打包格式。建议使用zip包，打包后目录结构如下



上传到WEB-INF\deploy的应用可以不用配置JetspeedContainerServlet及其映射。Jetspeed会做校验，如果为空，会添加该servlet配置。

注意：

1. 部署到这两个目录下的portlet应用必须是war格式的，而decorator 则必须是jar或zip格式。

2) Portlet应用也就是web.xml的版本必须是2.3以上，默认为2.3。（见JetspeedWebApplicationRewriterFactory）

### 系统处理流程

流程图如下：

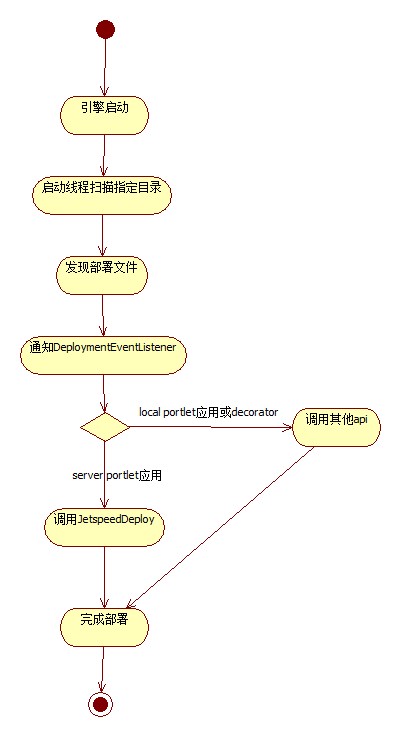


Figure 1 deploy 流程图

### 设计思想

1. 深入分析

deploy.xml配置文件配置了org.apache.jetspeed.components.ApplicationEventMethodsInvoker，当引擎启动后由spring启动DeploymentManager。DeploymentManager的工作思路是用一个线程定时扫描特定的部署目录，如果有部署文件的话，就通知DeploymentEventListener完成部署。

1. 部署过程：

Portlet：

DeploymentEventListener的实现类分DeployPortletAppEventListener和DeployDecoratorEventListener两大类。

对于servlet portlet应用，由DeployPortletAppEventListener调用JetspeedDeploy类完成部署。之后由servlet容器启动该应用，当应用启动之后，JetspeedContainerServlet会把该应用注册到jetspeed；

对于local portlet应用，由DeployPortletAppEventListener完成把文件copy到WEB-INF\apps 目录下，然后调用org.apache.jetspeed.tools.pamanager.PortletApplicationManagement注册应用。注意，对于local portlet没有servlet容器启动应用这一步。

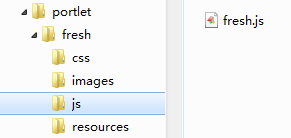
Decorator：

Decorator的部署主要是把文件copy到指定目录(decorations\fragmntypet\fragmentId\)并对国际化文件进行一些额外处理。

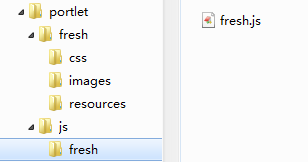
对国际化文件的处理。处理之前需要查找出国际化文件夹，而查找逻辑是根据文件夹名称长度为2确定该文件夹（如js）保存国际化资源的文件夹。然后把该文件夹及文件夹包含的文件，剪切到decorations\fragmntypet\js\decoratorId\下。

部署过程中，如果存在相同类型相同id的decorator，那么会卸载已存在的decorator。卸载只是删除了对应的decorator文件夹，以及部署时被移动的的国际化文件，如上面提到的decorations\fragmntypet\js\decoratorId文件夹。

部署前的文件目录结构，fresh.js是portlet/fresh/js文件夹下的文件。



部署后的文件目录结构，fresh.js是portlet/js/fresh下的文件。



相应代码：

DeployDecoratorEventListener.invokeDeploy(xxx)

DeployDecoratorEventListener.invokeUnDeploy(xxx)

1. 当存在同名已注册应用时，及portlet-app id花絮

Copy war包结束后，会调用PortletApplicationManagement.startLocalPortletApplication(xxx) 或PortletApplicationManagement.startPortletApplication(xxx) 来注册应用。如果是local portlet应用，会以文件名为portlet应用的id，如果是servlet portlet应用，则会以JetspeedContainerServlet的初始化参数contextName为portlet应用id。另外，如果web.xml没有配置JetspeedContainerServlet，而是由jetspeed在部署校验时添加，那么该contextName会被赋值为war包的文件名。

如果jetspeed中已经支持了同名的应用，jetspeed会尝试停止已经存在的应用。此时jetspeed会先判断工程类型是否一致（PortletApplication.WEBAPP 或PortletApplication.LOCAL）。如果工程类型不一致，则抛出异常，不会进行部署。如果是local portlet应用，jetspeed会以war的名称去查询已经注册的应用，如果是servlet portlet应用则以JetspeedContainerServlet的初始化参数contextName查询已经注册的应用。

综上，还可以得出一个结论。那就是，portlet.xml的<portlet-app>标签下的id没有被使用，并不需要指定该属性。经对servlet portlet应用的测试，删除该属性，可以正常启动和访问，数据显示的portlet-app id为JetspeedContainerServlet的contextName。

1. 可配置项

WEB-INF\deploy和WEB-INF\deploy\local目录的扫描间隔可修改override.properties下的autodeployment.delay项。

另外，请参考关键类DeployPortletAppEventListener的说明，附有多个关于目录的可配置项。

1. Local portlet 和 servlet portlet的区别。

功能上Local portlet只支持 action和render请求(见LocalPortletInvoker)。

另外，servlet 容器不知道 local portlet应用的存在，该应用被copy到WEB-INF\apps目录下，然后调用jetspeed的api注册应用。所以local portlet应用也不需要配置JetspeedContainerServlet。

servlet portlet就是我们常见的portlet，支持jsr168、jsr286标准。该应用会被copy到servlet容器可访问到的位置，然后由servlet容器启动该应用。最后由该应用web.xml中配置的JetspeedContainerServlet负责应用的注册。

相关代码请参考DeployPortletAppEventListener的deployLocalPortletApplication和deployPortletApplication方法，以及JetspeedPortletInvoker接口的两个实现类ServletPortletInvoker和LocalPortletInvoker。

1. 如何卸载（portlet and decorator）

Portlet，调用ApplicationServerManager的实现类，把应用从servlet容器卸载。但是卸载的portlet应用信息依然存储在jetspeed的数据库中。

Decorator，重新部署相同类型相同名称的decorator时，会自动卸载，然后在部署。Decorator的卸载只是简单的把和该decorator相关的文件从文件系统删除。调用DeployDecoratorEventListener.invokeUndeploy(xxx)方法即可。

1. 类图

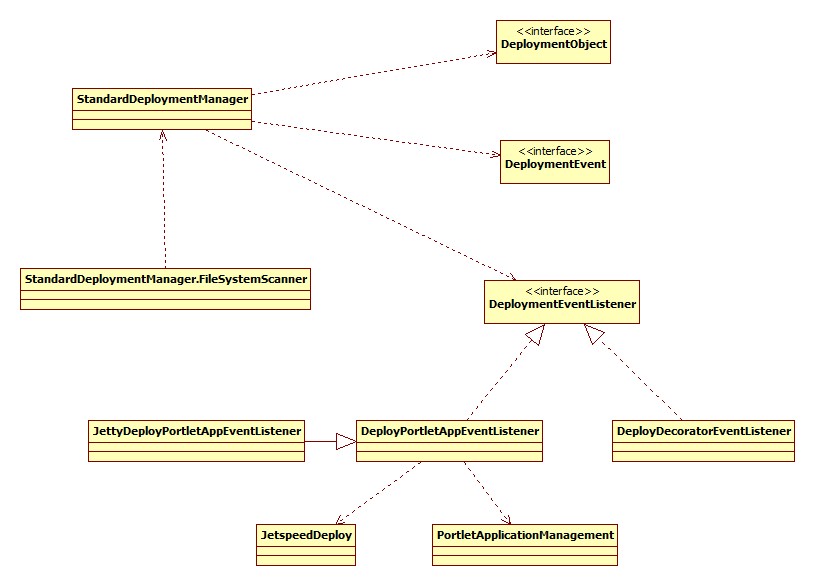


Figure 2 deploy类图1

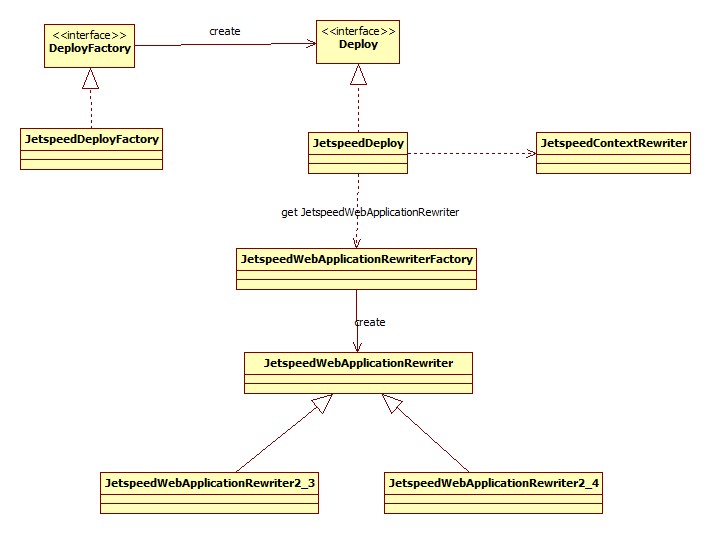


Figure 3 deploy类图2

关键类：

1. StandardDeploymentManager

该类的逻辑较简单，工作思路是用一个线程定时扫描特定的目录，如果有部署文件的话，就通知DeploymentEventListener完成部署。

1. DeployPortletAppEventListener。该类在deployment.xml中配置。从配置可以看出，关于部署目录都可以通过配置override.properties进行修改。

默认配置如下：

autodeployment.apps.internal.path=${applicationRoot}/WEB-INF/apps，localPortlet部署路径

autodeployment.target.dir=${applicationRoot}/../，servletPortlet部署路径

autodeployment.staging.dir=${applicationRoot}/WEB-INF/deploy，servletPortlet上传路径

autodeployment.apps.local.path=${autodeployment.staging.dir}/local，localPortlet上传路径

applicationRoot是org.apache.jetspeed.engine.JetspeedServlet的初始化参数，在web.xml中配置。

1. DeployDecoratorEventListener
2. JetspeedDeploy

该类负责copy web-inf/deploy目录下的servlet portlet应用到某指定目录，以便后续都servlet容器启动该应用。

在copy之前，会对web.xml、portlet.xml、context.xml、portlet.tld、portlet\_2\_0.tld等文件进行校验。其中web.xml、portlet.xml不能为空。portlet.tld、portlet\_2\_0.tld如果为空，则复制/org/apache/jetspeed/tools/deploy/目录下相应的文件到工程目录下。此外，还会有格式的校验及JetspeedContainerServlet补全等。

1. ApplicationServerManager

实现该接口，调用servlet容器的接口，以便支持servlet portlet应用的部署，卸载，启动，停止。这里的所有操作都是对servlet容器而言的，不包括注册到jetspeed和从jetspeed去除该应用的注册信息的操作。

### 重要数据模型

### 参考资料

### 问题记录

1. 同id的decorator部署之后，文件copy过去了，但是并没有清空之前的配置文件缓存，并从新加载。导致新的配置修改不起作用。
2. 我们的layout工程可以考虑部署为local portlet的形式，local portlet 应该会比 servlet portlet有更好的性能。(未求证)

## jetspeed-file-cache

### 设计概述

1、模块依赖

该模块被jetspeed-page-manager所依赖。

2、整体功能描述

配置文件配置了两个org.apache.jetspeed.cache.file.FileCache实例，分别是cache.xml定义的PageFileCache，importer-page-manager.xml定义的ImportPageFileCache。

PageFileCache被page-manager.xml所引用，ImportPageFileCache被importer-page-manager.xm（也就是定义了该bean的配置文件）所引用。

经debug，folder、page、link导入导出时会使用到这FileCache缓存，但不清楚是否其他情况下也会使用该缓存。

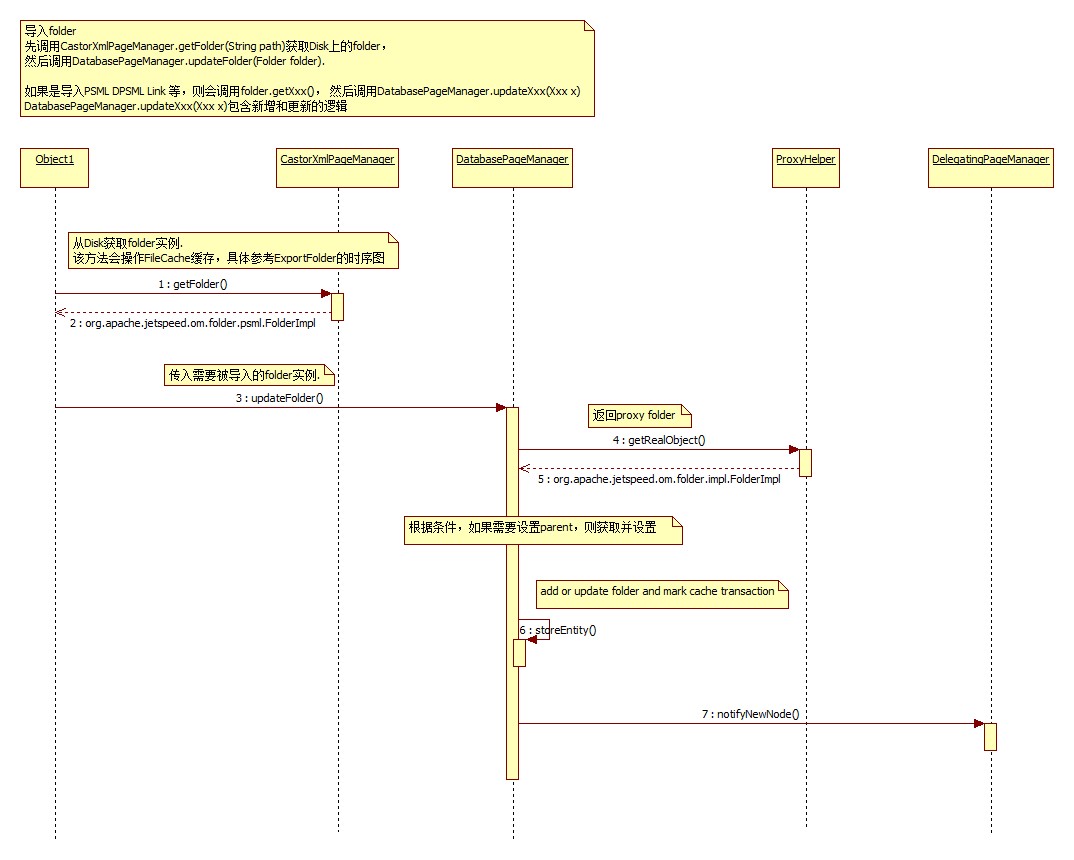
### 系统处理流程

下面以导入导出folder为例进行说明（导出其他文件类似，时序图内有注释）。

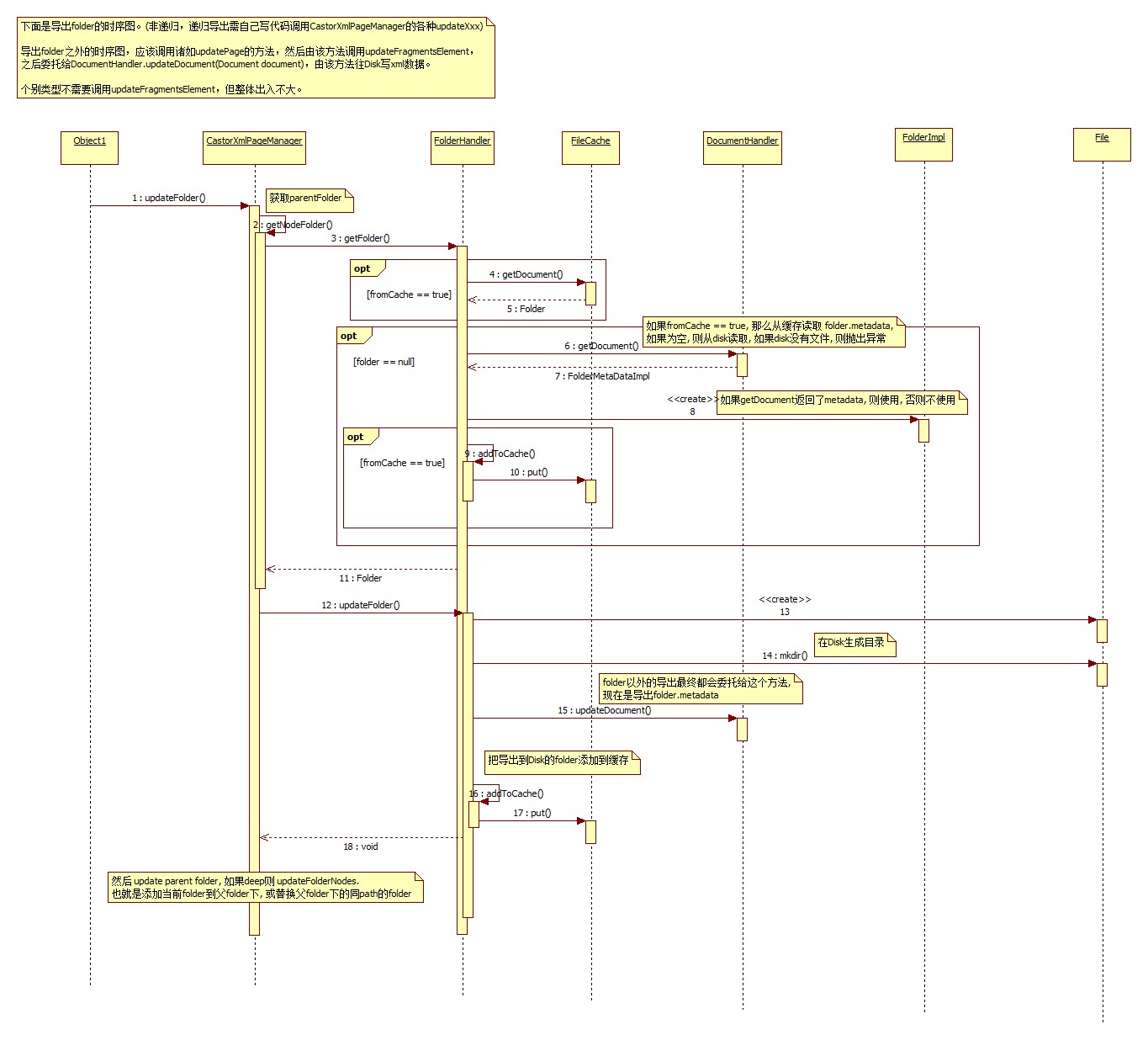
1. 导入的时序图画得比较简单。先调用CastorXmlPageManager.getFolder(String folderPath)，根据Disk的folder.metadata构建folder实例。然后调用DatabasePageManager.updateFolder(Folderfolder)。完成导入。

如果是导入PSML DPSML Link 等，则会调用folder.getXxx()获取相应的实例，然后调用DatabasePageManager.updateXxx(Xxx x)

DatabasePageManager.updateXxx(Xxx x)包含新增和更新的逻辑



1. 导出时，调用CastorXmlPageManager的updateXxx方法，Xxx对应Folder、Page、PageTemplate、Link、FragmentDefinition、DynamicPage等。然后委托FolderHandler或DocumentHandler对应的方法进行缓存操作和Disk文件读写。

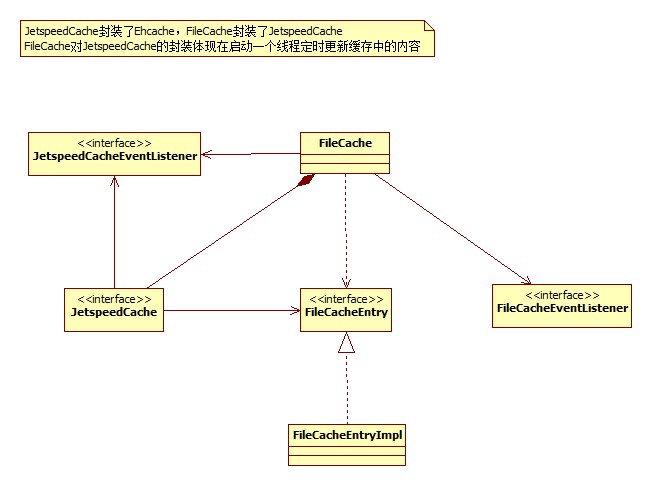


### 设计思想

1、深入分析

导入导出进行缓存时，会以node.getPath()为key进行缓存，所以给每个需要导入导出的node设置path时应该一致。所谓一致是指，给父把folder的path设置成“/sample//super”，然后子目录的path设置成“/sample/super/sub”，在导入导出的逻辑中需要查找其父folder，那么这个时候从缓存读取“/sample/super”为空，因为父folder在缓存中的key为“/sample//super”。如此便会导致再次构建父folder，并以“/sample/super”为key加入缓存。

2、类图



### 重要数据模型

### 参考资料

### 问题记录

## jetspeed-header-resource

### 设计概述

1、模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

2、关键技术、核心算法、思想

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-id-generator

### 设计概述

该模块目前只有一个类。该类唯一的功能是返回一个唯一的String标识。

该类被AbstractBaseFragmentElement和AbstractPageManager所调用。

## jetspeed-locator

### 基本设计概述

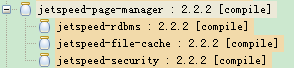
jetspeed-locator模块只被jetspeed-profiler所依赖。Jetspeed资源（page、folder、menu、link）查找有两种方式，一是profiler，而是通过简单的算法映射URL。具体来说只有当jetspeed通过profiler查找资源时才会依赖jetspeed-locator模块。

[请参考jetspeed-profiler模块的相关资料。](#_jetspeed-profiler)

## jetspeed-page-manager

### 设计概述

1、模块依赖



2、整体功能描述

本模块用于对数据库或者psml文件中的页面、目录、链接的增删查改操作，也会对页面、目录、链接、portlet进行缓存操作。

### 系统处理流程

本条描述模块的主要处理流程，

1、获取目录处理流程

流程图



### 设计思想

1、深入分析

本模块用于门户中页面，目录，连接，portlet，及其权限的增删查改操作。而且获取资源时，Jetspeed使用了OJB的缓存管理。当通过OJB查询数据库时，而且是通过identity查询时，就会自动调用ObjectCache的cache方法缓存对象。

OJB提供缓存接口ObjectCache，页面缓存DatabasePageManagerCache实现该接口

Java代码

public interface ObjectCache

{

/\*\*

\* Write to cache.

\*/

public void cache(Identity oid, Object obj);

/\*\*

\* Lookup object from cache.

\*/

public Object lookup(Identity oid);

/\*\*

\* Removes an Object from the cache.

\*/

public void remove(Identity oid);

/\*\*

\* Clear the ObjectCache.

\*/

public void clear();

}

在ORM映射中配置如下：

<class-descriptor

class="org.apache.jetspeed.om.folder.impl.FolderImpl"

table="FOLDER"

>

<object-cache class="org.apache.jetspeed.page.impl.DatabasePageManagerCache">

</object-cache>

...

</class-descriptor>

以获取目录为例，前面的流程图展示了具体的过程，具体步骤为：

* + 1. 根据目录的路径获取目录
    2. 如果缓存中查询目录，如果存在就返回Folder，如不存在就进行第3步
    3. 通过OJB查询数据库，并且通过OJB缓存接口建立缓存

2、关键技术、核心算法、思想

3、类图

DatabasePageManager负责对数据库的页面、目录、链接进行增删查改



CastorXmlPageManage负责对psml文件中的页面、目录、链接进行增删查改

DatabasePageManagerCache针对页面目录等资源增删查改操作时建立缓存管理



### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-portal

### 设计概述

1、模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. Aggregator(聚合渲染)

Portal提供将多种内容源以及Portlet在一个浏览器中以一个整合的页面展示。这个将多种内容聚合和渲染到一起的过程被称作聚合。在Jetsepeed中，聚合模块被封装成可插拔的Spring组件插到Jetspeed引擎中。

Jetspeed中支持两种聚合渲染的方式，一种是串行聚合渲染，另一种是并行聚合渲染。

串行聚合渲染可分为对页面（page）的聚合渲染，以及对Portlet的聚合渲染。对于页面的聚合渲染，主要是去渲染一个页面所有的Portlet来创建页面内容。而对于Portlet的聚合渲染，是针对单个Portlet的渲染去创建Portlet内容。主要的相关类是“PageAggregator”与“PortletAggregator”。

并行聚合渲染主要是针对页面（page），是用并行的方式渲染整个页面的Portlet，从而组织页面内容。页面上每一个Portlet都在它自己的线程中进行渲染。与此同时，在此模块有一个workManager去管理线程池并同步工作线程。

1. 功能点2

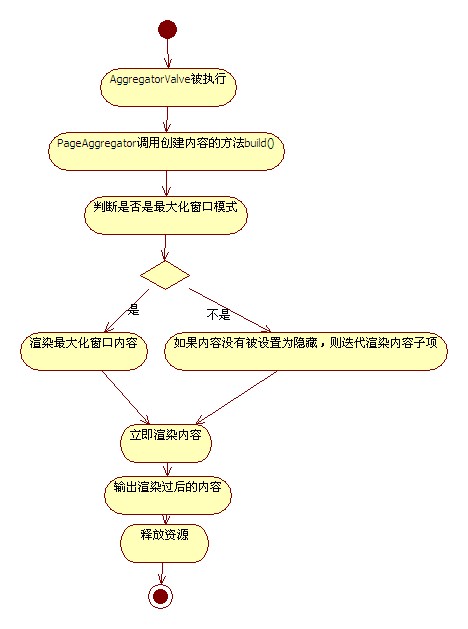
3、与其他模块的关联（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

### 系统处理流程

本条描述模块的主要处理流程，

1. 页面（page）聚合渲染处理流程
2. 串行聚合渲染



1. 并行渲染

（由于存在多种分支处理，处理流程图会较难展示。详情请见 设计思想）

2、xx处理流程

### 设计思想

1. 关键技术、核心算法、思想
2. 并行渲染。

并行渲染是针对页面的，当渲染器的配置中选用了并行渲染（即在aggregator.xml中使用的是AsyncPageAggregatorImpl）。则当渲染阀门被执行的时候，就会调用并行渲染类的创建方法（build）。与串行相同，渲染会先去查看当前窗口的状态，如果是最大化状态，则会被立即渲染，与串行渲染没有不同。而当此时不是最大化状态，则与串行渲染的区别就显现出来了。由于整个内容时由布局和Portlet组成，组成了一个树状结构的模型，若此时的元素是布局（layout），会迭代此布局下的元素，并将布局加入集合储存起来。如果此时的元素是Portlet，则会创建Portlet单独的Portlet渲染线程（job），并根据其超时时间（timeout）大于0储存到并行线程的集合中，其他储存到串行线程的集合中。此超时时间在jetspeed-portlet.xml中配置，若没有配置，会去查找是否有默认设置（在jetspeed.properties，或者overrider.properties中设置）。

对于串行线程集合里的线程，当执行渲染时，便直接运行该线程。对于并行线程集合里的渲染线程，执行渲染时，会由WorkerMonitor去管理，此时如果有可用的work，则会将渲染线程（job）加入work中，执行该线程。而如果当前没有空闲的work，就会将其放置到一个FIFO序列（queue）中，当出现空闲的work时，会从此序列中POP出来，再将其加入到可用的work里。

之后，当前主线程会等待所有并行线程完成后，再继续往下走。接着立即渲染之前加入布局集合中的所有布局。最后立即渲染最后的根元素。完成整个渲染步骤。

1. 类图
2. 并行渲染主要类：AsyncPageAggregatorImpl、PortletRendererImpl、WorkerMonitorImpl

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-portal-site

### 设计概述

1、模块依赖

从POM文件来看，jetspeed-portal-site是一个独立的模块。对应的spring配置文件时 page-manager.xml。

2、整体功能描述

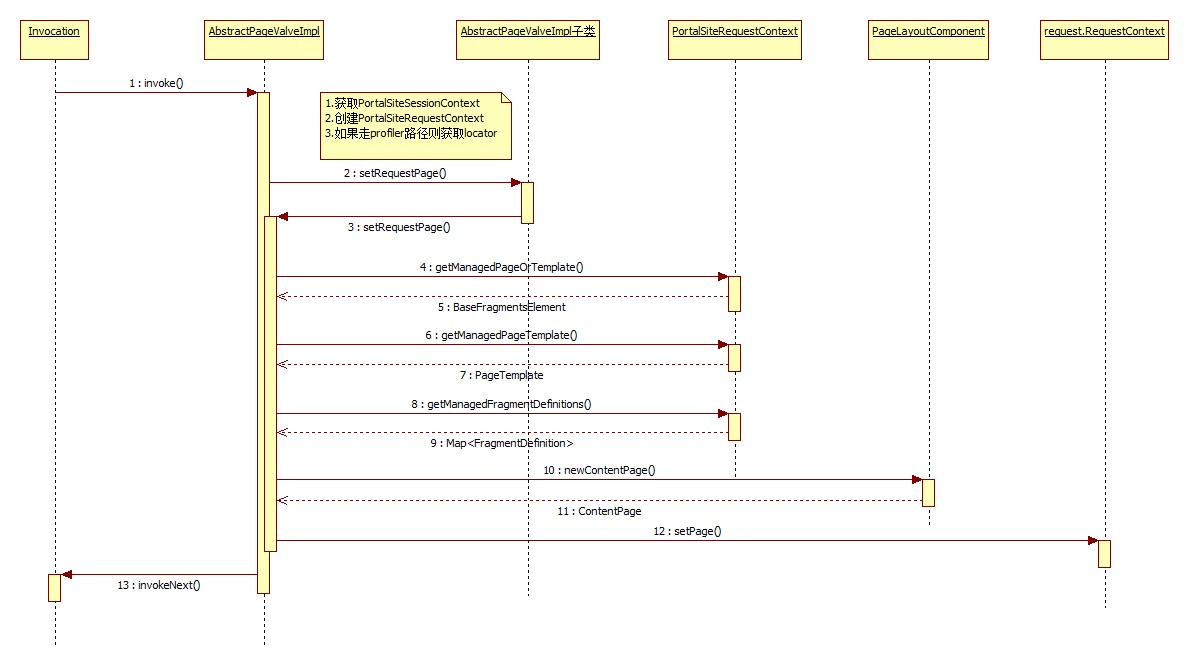
Portal-site模块专门用来描述jetspeed站点。初步了解，有两个重要的功能。

* 1. 提供站点的逻辑视图，通过视图定位资源（page、folder、link、menu）。
  2. 返回菜单供主题使用

### 系统处理流程

本条描述模块的主要处理流程，

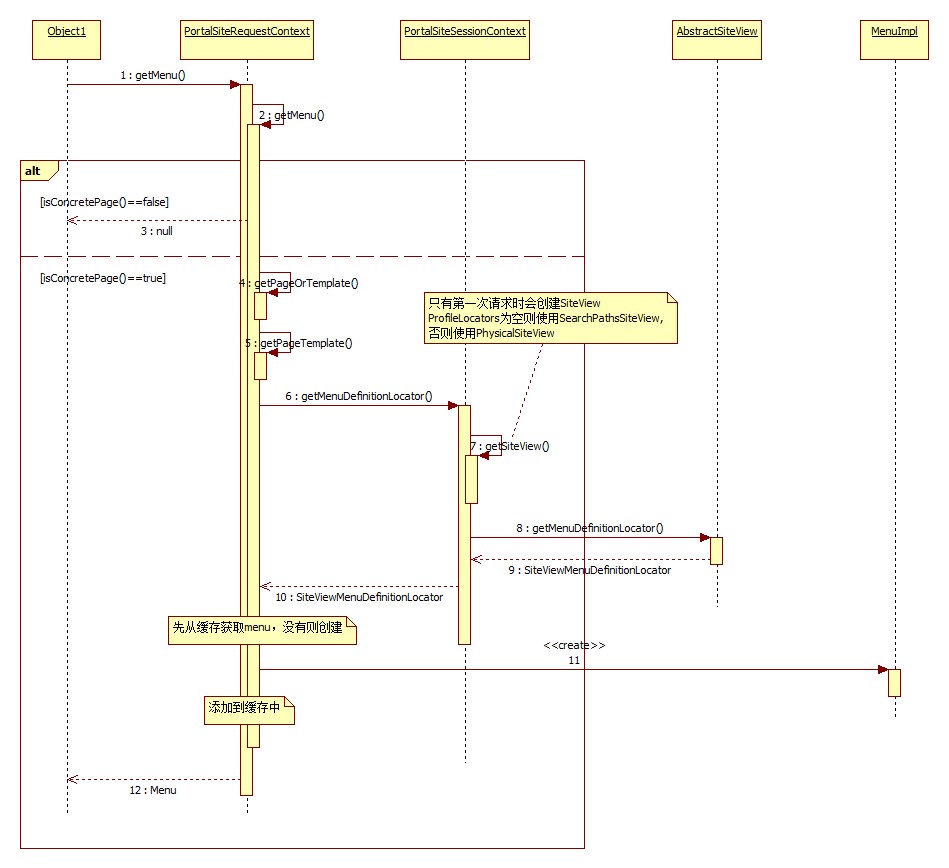
1. 定位页面及其他资源



Profiler模块调用PortalSiteSessionContext创建了PortalSiteRequestContext，然后通过PortalSiteRequestContext进行页面定位。PortalSiteRequestContext在定位页面时把操作委托给PortalSiteSessionContext。第一次定位资源时，PortalSiteSessionContext会根据当前请求参数创建AbstractSiteView（如果profilerLocator不为空则创建SearchPathsSiteView，否则创建PhysicalSiteView）。之后的资源定位都会调用所创建的AbstractSiteView实例。

另外SearchPathsSiteView依赖当前用户的所有profilerLocator，所以当用户的profilerLocator有变化时，需要更新PortalSiteSessionContext。

2、获取菜单



在创建装饰和页面聚合的时候需要获取菜单。这时候会调用PortalSiteRequestContext的getMenu方法，该方法委托给PortalSiteSessionContext，PortalSiteSessionContext最后委托给AbstractSiteView，AbstractSiteView最终返回SiteViewMenuDefinitionLocator，PortalSiteRequestContext根据该返回值从缓存中获取menu或者创建一个menu实例。

### 设计思想

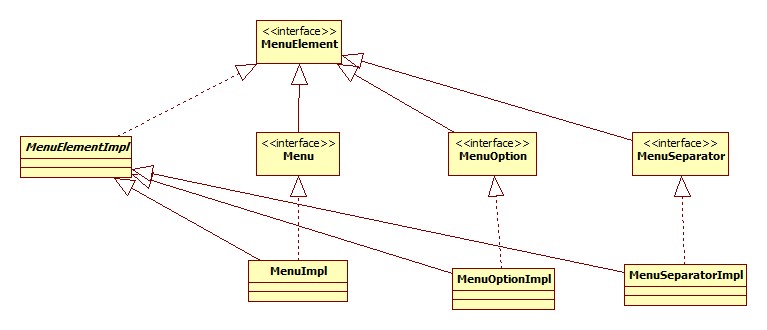
1、深入分析

1）缓存

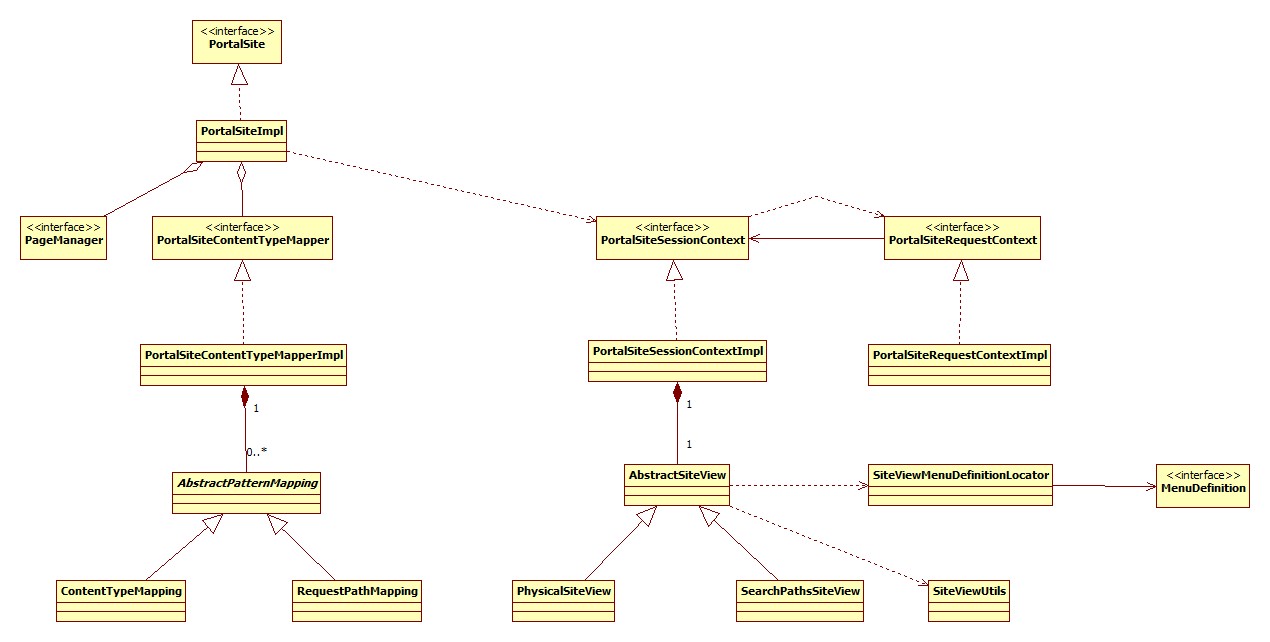
2、关键技术、核心算法、思想

3、类图

1. 菜单



1. portal site



3）关键类，及其作用

（1）PortalSiteSessionContextImpl

这里使用实现类而不是接口，是因为实现类实现了多个接口，并且提供了很多额外的方法。PortalSiteSessionContextImpl也是直接依赖实现类，而不是接口。

这个类的存在有两个目的，缓存AbstractSiteView和记录用户访问的资源历史（包括缓存被用户访问过的页面等资源）。所以PortalSiteSessionContext里面的资源都不需要被序列化，因而都被声明成transient的。

（2）PortalSiteRequestContext

每一个请求都会创建一个新的实例。该类的很多操作都是委托PortalSiteSessionContextImpl完成的。

（3）AbstractSiteView及其子类

对应一个PortalSiteSessionContext只有一个实例。（请参考PortalSiteSessionContext的getSiteView方法）。该类有两个主要功能，定位资源和获取菜单。

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计

|  |  |
| --- | --- |
| folder\_menu | Folder节点上定义的menu |
| folder\_menu\_metadata |  |
| page\_menu | 覆盖同名称的folder\_menu |
| page\_menu\_metadata |  |

### 参考资料

### 问题记录

*1.对于portal-site模块org.apache.jetspeed.om.folder.proxy和org.apache.jetspeed.om.page.proxy和org.apache.jetspeed.page.document.proxy包下的类，其用途还不是很清楚。猜测是对用户访问多的页面等资源进行了代理和缓存。如果这样被缓存的资源的本体有更新的话，那么也要更新这样缓存。*

## jetspeed-portlet-factory

### 设计概述

1、模块依赖



此模块并不依赖其他模块。jetspeed-registry模块会依赖该模块。jetspeed-portal会依赖factory模块的api，从而调用到该模块，并不直接依赖。

2、整体功能描述

整体描述

本模块主要包括将Portlet应用注册到factory中，在factory中注销Portlet应用，获取portlet实例等一系列运行时操作。实例信息等缓存在内存中，当应用被卸载，则销毁内存中的Portlet相关对象。分别提供了针对Portlet应用和Portlet定义的接口，这些接口在其他模块被调用，接口之间相对独立。

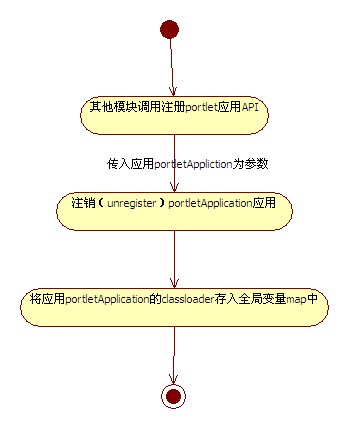
1. Portlet应用
2. Portlet应用注册、注销、是否已注册。
3. 获取Portlet应用的类加载器(classloader)。
4. 加载获取、重加载Portlet应用的资源。
5. 获取Portlet应用的所有监听器。
6. 根据过滤器名称获取Portlet应用的该过滤器。
7. Portlet定义
8. Portlet实例的获取。
9. 获取Portlet偏好的校验器。
10. 加载获取、重加载Portlet定义的资源
11. Portlet是否包含help模式

### 系统处理流程

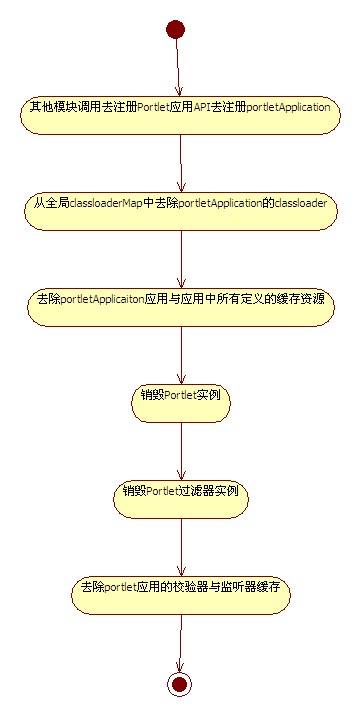
本条描述模块的主要处理流程，

1、Portlet应用注册与注销

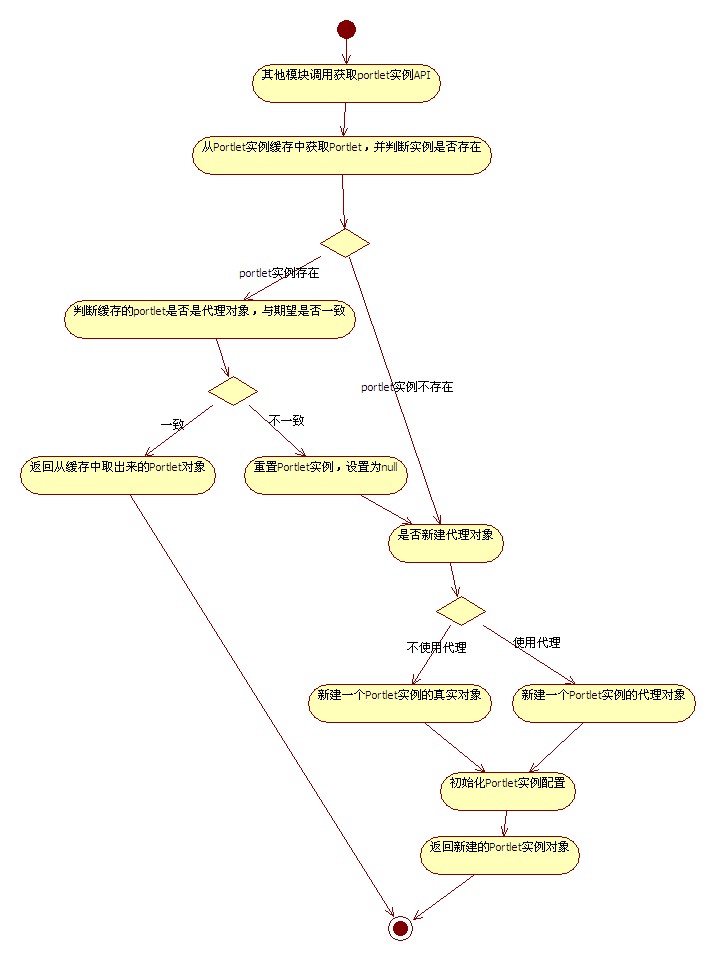
1）注册Portlet应用



2）注销Portlet应用



2、Portlet实例获取



### 设计思想

1、深入分析

此模块普遍采用了缓存处理，对于PortletApplication的注册，首先是注销（unregister）该应用，然后将该应用的classLoader存入一个Map中。注册主要在应用被部署的时候被调用。对于其他接口的实现都依附于该classloader，应用反射的机制获取相关处理类的实例，并且在处理Portlet实例相关的创建、销毁、初始化，需要设置当前的线程上下文的classloader为该应用的classloader进行类的加载，完成操作后，又设置回之前的classloader。另外，对于一个portlet应用是否注册，也是判断classLoader的Map对象中是否包含该应用的classloader。

对于注销(unregister)，首先将classloader从Map中删除，然后需要清除和该应用相关的所有的缓存，并销毁portlet实例对象。

对于获取Portlet实例，可根据条件选择是否使用代理，从而获取代理对象或真实的Portlet实例对象。该接口在ServletPortletInvoker、LocalPortletInvoker中被调用，在这两个类中被调用的主要区别在于，LocalPortlet需要使用该portlet应用自己的classloader去加载类，而ServletPortlet不用特意去设置。

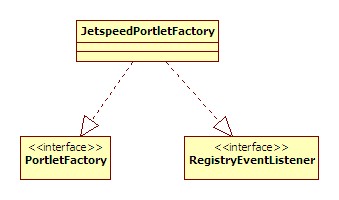
Portlet资源的获取，Portlet应用的资源与Portlet定义的资源是分开的。对于Portlet应用主要是根据Portlet应用的资源模块去加载，而对于Portlet定义主要加载三个信息，即title、short-title、keywords，根据不同的语言获取不同语言的结果。

2、关键技术、核心算法、思想

此模块比较频繁的使用了类的反射机制加载类，线程上下文类加载器设置，多线程synchronized()同步代码块、以及缓存等技术。

3、类图

此模块重要的类：JetspeedPortletFactory，主要包含了对Portlet应用的一系列操作，外部模块主要调用该类的接口进行处理。



### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

本模块不直接与数据库交互。

### 参考资料

1、wiki上的记录

2、官网链接页面

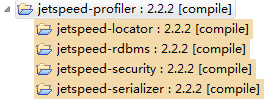
### 问题记录

## jetspeed-profiler(及space)

### 设计概述

1、模块依赖

Profiler模块依赖



2、整体功能描述

Jetspeed资源定位方式有两种，分别是profiler和space方式，由PageValveImpl和ProfilerValveImpl实现。对于这两种实现方式的详细分析请阅读参考资料。

[http://svn.apusic.net/aps/doc/trunk /设计管理/设计文档/jetspeed/profiler](http://svn.apusic.net/aps/doc/trunk%20/设计管理/设计文档/jetspeed/profiler)下有详细的profiler分析报告

[http://svn.apusic.net/aps/doc/trunk /设计管理/设计文档/jetspeed/](http://svn.apusic.net/aps/doc/trunk%20/设计管理/设计文档/jetspeed/space)space下有详细的space分析报告

<http://portals.apache.org/jetspeed-2/devguide/guide-profiler.html>官方文档

<http://portals.apache.org/jetspeed-2/deployguide/guide-jetui.html#spaces>官方文档

切换profiler和space：

配置override.properties(jetspeed.properties)文件的jetui.customization.method的值为 server，则使用profiler进行资源定位，否则使用space方式。

当用户访问jetspeed时，以下资源，PSML、folder、menu、link都由profiler或space进行定位。Profiler或space的定位逻辑总是在登录逻辑之后（如果有的话），在页面的渲染和聚合之前被调用。

3、相关valve：

PageProfilerValveImpl jetspeed默认的profiler valve，这个valve并不处理资源定位，而是决定转发到PageValveImpl或ProfilerValveImpl。

PageValveImpl jetspeed定位资源的两种方式之一，尚不清楚，有待跟踪。

ProfilerValveImpl jetspeed定位资源的两种方式之一，通过locator定位资源。

RefreshUserHomepageValveImpl已废弃。用户第一次登陆的时候，Copy用户模板目录下的文件到用户目录下。

CreateUserTemplatePagesValveImpl功能和RefreshUserHomepageValveImpl类似，但更强大。请阅读方法updateUserHomepagesByTemplate。

代码入口：

pipeline.xml里配置的pageProfilerValve，默认对应类org.apache.jetspeed.pipeline.valve.impl.PageProfilerValveImpl。

### 问题记录

## jetspeed-rdbms

### 设计概述

1、模块依赖



page-manager、security等模块依赖本模块定义的基类进行数据库的增删查改操作。

2、整体功能描述

本模块用于连接数据库和配置数据源

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

无

### 设计思想

1、深入分析

2、关键技术、核心算法、思想

3、类图

InitablePersistenceBrokerDaoSupport：DAO基类，init方法会默认加载OJB的OR映射文件，以后扩展APS的DAO类时，都应该继承这个基类



DBCPDatasourceComponent：支持dbcp的数据源组件

ConnectionManagerImpl：这个连接管理类，代替了OJB的默认ConnectionManagerImpl。它们的唯一区别是，jetspeed的ConnectionManagerImpl使用了自定义的方式获取连接工厂connection factory。

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

（无）

### 参考资料

1、wiki上的记录

2、官网链接页面

*http://portals.apache.org/jetspeed-2/devguide/dev-rdbms.html*

### 问题记录

1、SchemaAwareDataSourceProxy在api中有说明：它代理了数据源，而且在每次

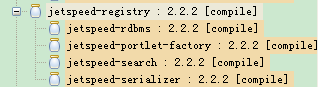
getConnection()调用时，都会注入sql查询，但是**为什么Jetspeed要这么做**呢？

2、SchemaAwareDataSourceProxy在jetspeed启动时并没有加载这个bean，而且在jetspeed.properties文件中并没有设置变量org.apache.jetspeed.database.set.schema.sql，那么这个bean中的**schemaSql应该设置什么值**？

## jetspeed-registry

### 设计概述

1、模块依赖

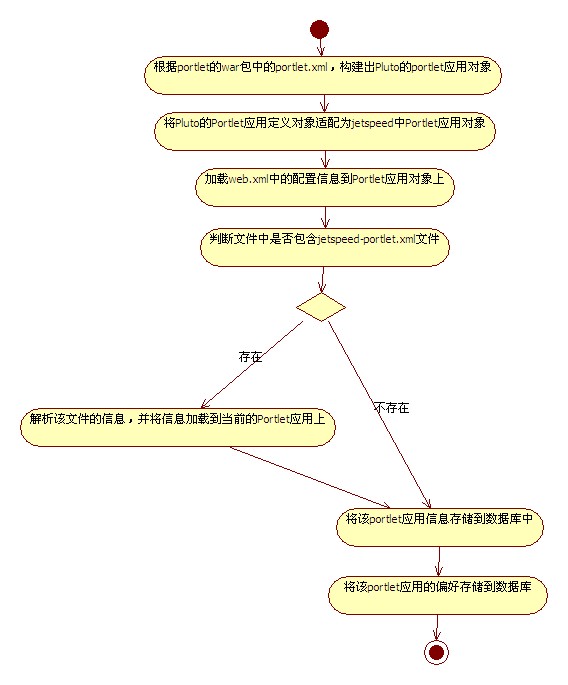


2、整体功能描述

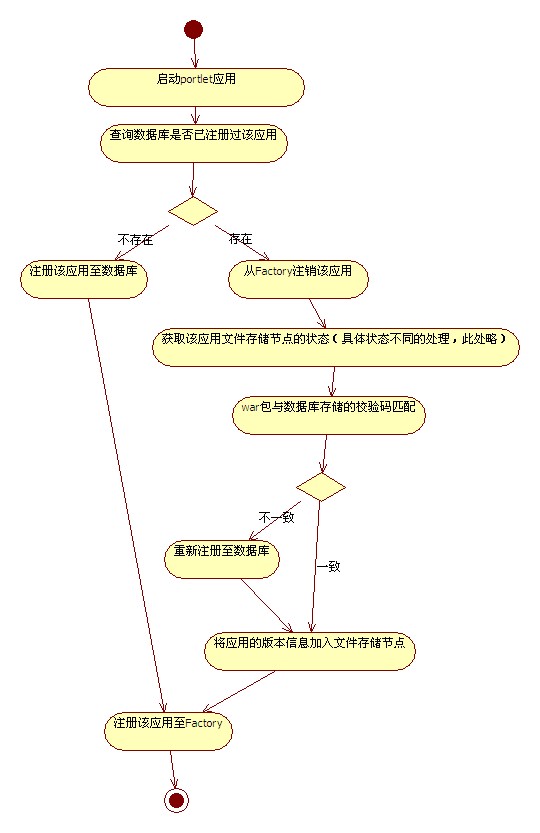
本模块负责将portlet应用和定义信息注册进数据库，从数据库中注销，以及获取portlet定义等操作。其中包含了对Portlet应用和Portlet定义映射数据库的数据模型以及数据库操作。并单独提供了偏好（preferences）映射数据的数据模型，以及数据库操作相关的服务。此模块还包含了jetspeed-portlet.xml文件的对应数据模型，以及对portlet应用相关配置文件的解析。

### 系统处理流程

1、Portlet应用注册到数据库流程



2、与Portlet注册相关的启动Portlet应用流程



### 设计思想

1、深入分析

此模块主要包含了对Portlet应用与定义相关的数据库的相关CRUD操作。主要关联Portlet应用的部署启动、应用的卸载，以及需要从数据库中查询Portlet定义的相关模块。此模块应用了OJB的持久化框架去管理相关的数据库操作，故而此模块包含了Portlet应用的OJB映射对象，以及OJB的相关配置文件。Jetspeed引入了jetspeed-portlet.xml文件，用于配置一些不同于其他Portal的Jetspeed特殊特性，例如并行渲染等。同时此模块使用jaxb技术对jetspeed-portlet.xml文件进行解析。

2、关键技术、核心算法、思想

OJB持久化框架，jaxb解析XML框架，XPath，多线程同步，缓存技术等。

3、类图

* + 1. Portlet应用与定义的CRUD操作类



2）Portlet应用偏好的CRUD操作类



3）包含配置文件的解析加载，以及将pluto的应用对象适配至jetspeed应用对象



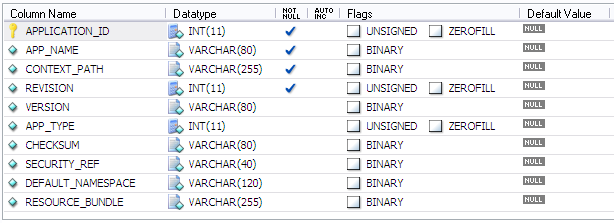
### 重要数据模型

1、以数据结构图描述相互关系

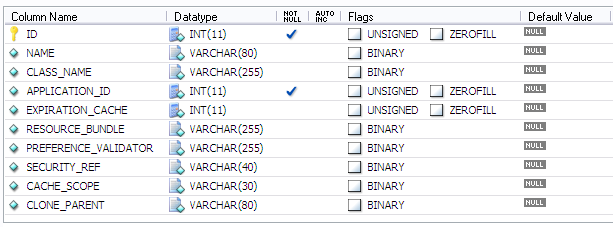
2、数据结构设计。

此模块涉及到portlet应用的所有相关表，不一一罗列，详细内容可参考ojb框架的配置文件：registry\_repository.xml以及与偏好相关的fast\_preferences.xml。其中覆盖了与Portlet应用相关所有的表定义信息以及关联关系。此处只列出较为重要的几个表：

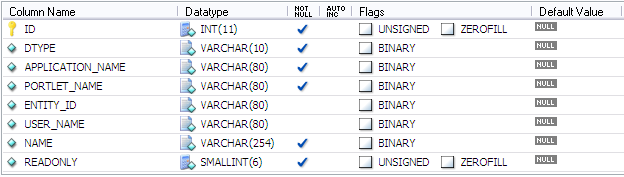
* + 1. PORTLET\_APPLICATION:



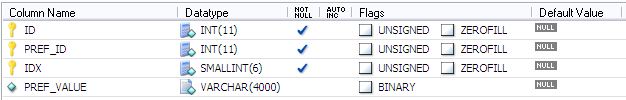
* + 1. PORTLET\_DEFINITION



3）PORTLET\_PREFERENCE



4）PORTLET\_PREFERENCE\_VALUE



其中PORTLET\_PREFERENCE\_VALUE包含外键PREF\_ID，对应PORTLET\_PREFERENCE表中的ID。

### 参考资料

1. wiki上的记录

<http://wiki.apusic.net/pages/viewpage.action?pageId=11340122>

2、官网链接页面

<http://portals.apache.org/jetspeed-2/deployguide/guide-registry.html>

### 问题记录

## jetspeed-search

### 设计概述

1、模块依赖

该模块是一个独立的模块，被jetspeed-registry模块依赖。

2、整体功能描述

Jetspeed-search模块提供对Jetspeed某些资源提供了搜索功能。默认支持对portlet应用和portlet定义的搜索。当portlet注册到jetspeed时，会调用searchEngine.add(pa) 方法和 searchEngine.add(pa.getPortletDefinitions())方法来更新索引。

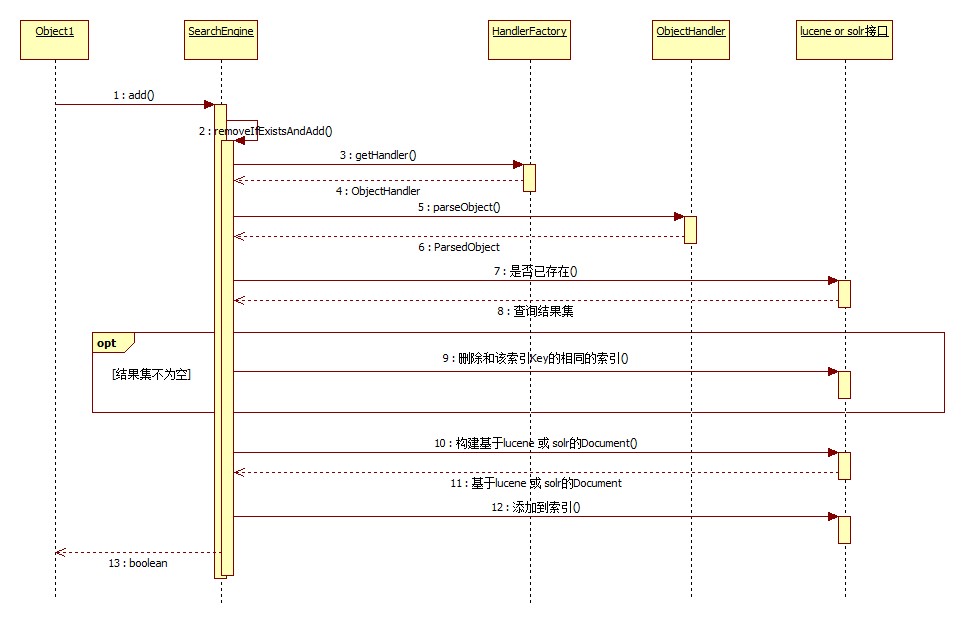
jetspeed支持通过lucene或solr来搜索，默认使用lucene。

注册portlet时会创建索引，请调试PortletApplicationManager.attemptStartPA(xxx…)

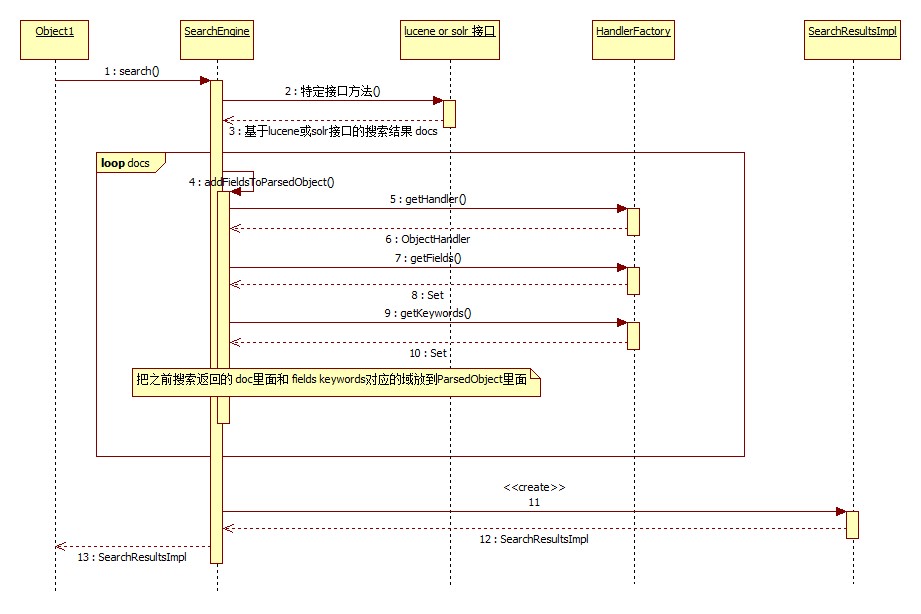
### 系统处理流程

本条描述模块的主要处理流程，

1. 添加索引（更新和删除类似）



1. 搜索



### 设计思想

1. 深入分析

操作索引或搜索都是通过调用SearchEngine的接口来进行。操作索引时，所有输入都会被转换成ParsedObject。搜索结果返回之后，SearchEngine会把搜索结果解析成ParsedObject的集合，最后封装成SearchResults。

把输入转换成ParsedObject(操作索引时)是通过ObjectHandler的实现类完成的。需要注意的是每对一种资源进行索引就需要提供一个ObjectHandler的实现类。

* 1. 扩展。如果需要搜索其他资源，只需要实现相应的ObjectHandler，把目标资源转换为ParsedObject。然后就可以把目标资源添加到索引。
  2. 切换solr。Solr支持内嵌或http模式。

(1) 如果使用内嵌模式，只需要把alternate/ search-embedded-solr-override.xml文件复制到override目录下即可。

(2) 如果使用http模式，

1. 部署solr服务。

复制 web-inf/solr/solr.xml文件到example/solr目录下(覆盖)

在example/solr下新建js文件夹

复制 web-inf/solr/js/schema.xml 和web-inf/solr/js/solrconfig.xml到example/solr/js下(覆盖)

1. 配置override.properties(jetspeed.properties)下search.http.solr.url配置项
2. 把alternate/search-http-solr-override.xml文件复制到override目录下

相关配置：

jetspeed.properties有若干关于搜索的配置项，搜索”search”即可查找到。

assembly/search.xml配置了HandlerFactory和SearchEngine类，其他类都通过这两个类实例化。

如果使用solr则还有较多相关的配置文件：

search-embedded-solr-override.xml配置了基于EmbeddedSolrServer的SearchEngine

search-http-solr-override.xml配置了基于CommonsHttpSolrServer的SearchEngine

web-inf/ solr文件夹下的配置文件：

web-inf/ sol/solr.xml配置Solr实例

web-inf/ sol/js/conf/admin-extra.html

web-inf/ sol/js/conf/elevate.xml

web-inf/ sol/js/conf/mapping-ISOLatin1Accent.txt

web-inf/ sol/js/conf/protwords.txt

web-inf/ sol/js/conf/schema.xml定义索引哪些字段，索引和查询时对字段做怎样的处理。

web-inf/ sol/js/conf/scripts.conf

web-inf/ sol/js/conf/solrconfig.xml对缓存等个性化配置，是系统全局性的配置；

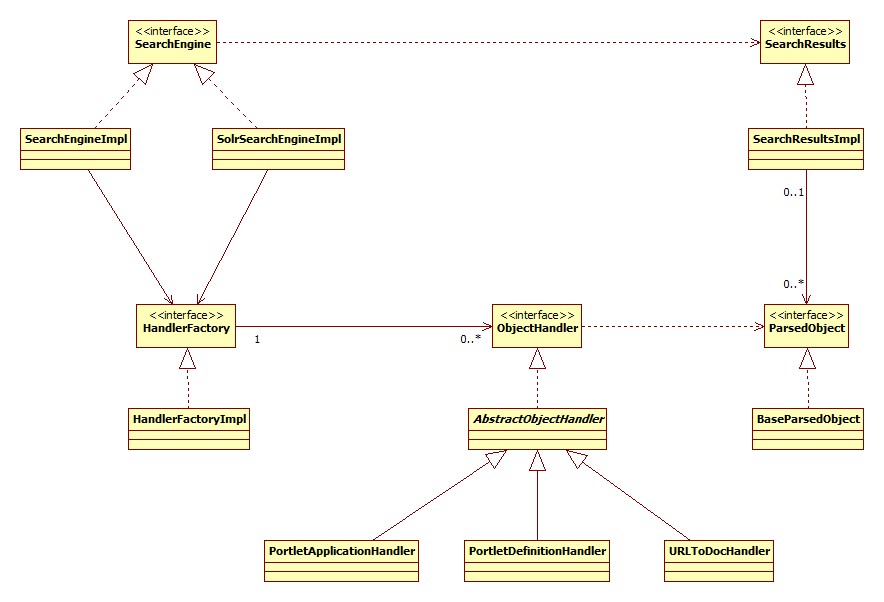
web-inf/ sol/js/conf/spellings.txt

web-inf/ sol/js/conf/stopwords.txt

web-inf/ sol/js/conf/synonyms.txt

web-inf/ sol/js/conf/xslt/\*.xml

2、类图



关键类说明

1）SearchEngine，提供把资源添加到索引，更新删除索引，搜索等接口。

2）ObjectHandler，是把各种资源转换为ParsedObject的助手类必须实现的接口。

3）ParsedObject，定义了jetspeed搜索支持的字段，这些字段会被加到org.apache.lucene.document.Document或者org.apache.solr.common.SolrInputDocument中。

4）SearchResults，查询返回的结果。

### 重要数据模型

### 参考资料

1、官网资料

<http://portals.apache.org/jetspeed-2/devguide/dev-search.html>jetspeed官网资料

<http://lucene.apache.org/core/> lucene官网

<http://wiki.apache.org/jakarta-lucene/FrontPage> lucene wiki

<http://wiki.apache.org/solr/> solr官网

### 问题记录

1. *lucene或solr那个更适合portal*
2. *Solr嵌入式和http模式的区别(使用方式和场景)*
3. *web-inf/ solr文件夹下的配置文件*

## jetspeed-serializer

### 设计概述

1、模块依赖



2、整体功能描述

本模块用于：种子数据的导入导出(种子数据：数据库中除去页面目录链接相关数据的剩余数据，如用户角色，portlet应用，定义，个性化规则、访问统计等数据)

### 系统处理流程

本条描述模块的主要处理流程，

1、导出种子处理流程

* 1. 流程图



2、种子数据导入处理流程

导入流程其实是导出流程的逆向过程，这里就不画流程图了

### 设计思想

1、深入分析

2、关键技术、核心算法、思想

本模块利用了xml解决跨数据库间的数据导入导出问题

1. 定义xml格式
2. 使用javolution将xml数据转换为java bean
3. 将java bean和OR映射成的数据模型进行转换，如JSRole---Role
4. 类图

JetspeedSerializerImpl：门户数据导入导出的核心类，也是总入口，它会调用各个子模块的导入导出功能。

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

本模块会对数据库所有表导出或者导入数据

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-security

### 设计概述

1、模块依赖



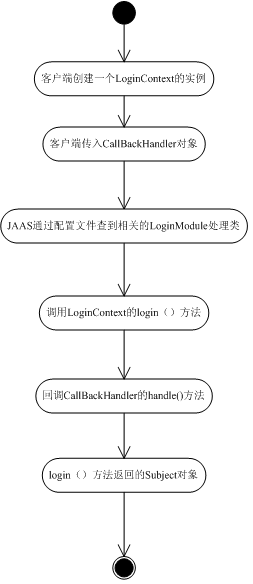
2、整体功能描述

本模块负责jespeed的验证和授权功能，很多其他模块都依赖于这个安全模块，而且本模块也提供了ldap集成的功能，ldap实现类会在后面介绍

### 系统处理流程

本条描述模块的主要处理流程，

1、 Jetspeed基于JAAS的验证流程

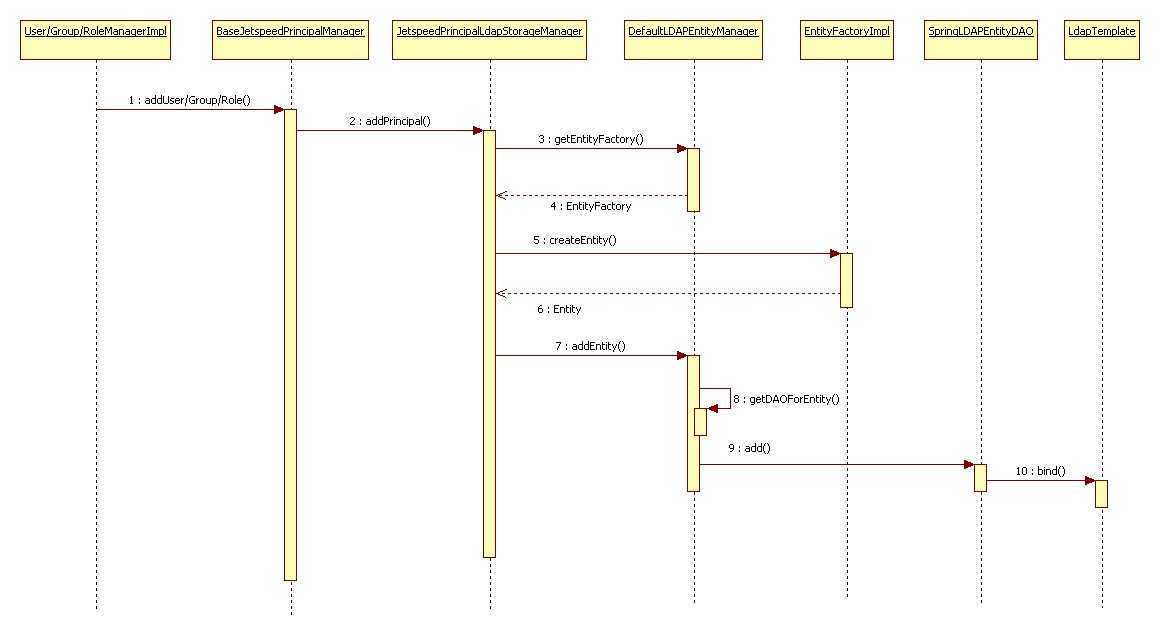


1. Jetspeed基于JAAS的授权流程

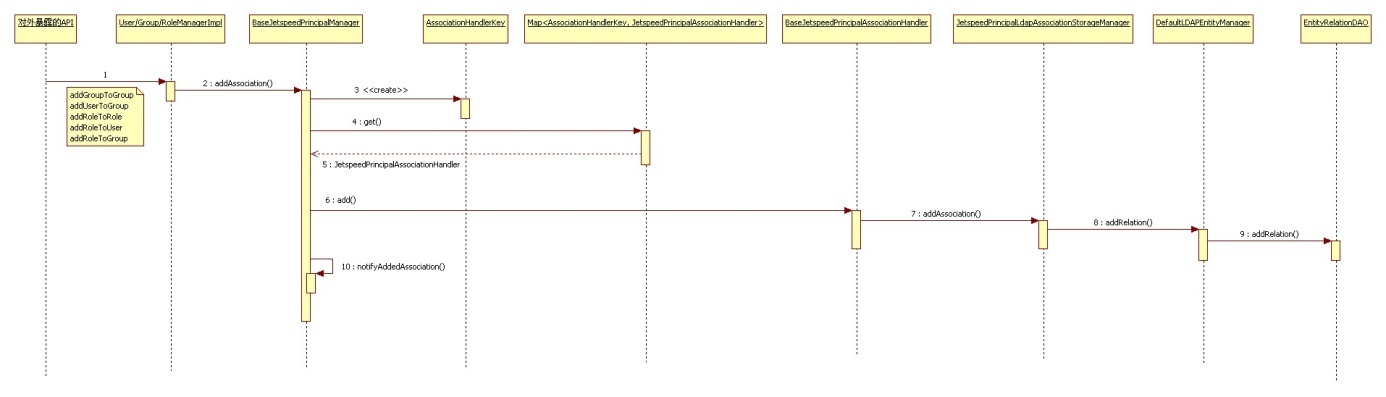
当通过Permission机制对页面上的portlet授权后，页面在渲染时，会判断当前用户是否拥有权限访问这个portlet。



1. Ldap中添加用户以及添加关联的时序图
2. 添加用户



2) 添加实体的关联关系



### 设计思想

1、深入分析

1) Jetspeed安全模块

Jetspeed的安全模块，建立在JAAS之上，实现了验证和授权两大模块。建议学习本模块前，需要结合JAAS的知识来理解。

1. Jaas的验证流程

Jetspeed基于JAAS的验证流程，如流程图所以，Jetspeed的验证过程是标准的jaas验证，只不过Jetspeed实现了自己的loginModule：org.apache.jetspeed.security.impl.DefaultLoginModule

1. Jetspeed基于JAAS的授权流程

Jetspeed的permission授权机制，本质上也使用了Jaas的授权机制，以页面渲染portlet为例子，JAAS权限检查的具体步骤如下：

* + 1. 引擎调用PageAggregator.aggregateAndRender()渲染portlet
    2. 检查portlet权限SecurityUtil.checkPortletAccess(String portletUniqueName)
    3. 通过Jaas的机制判断portlet权限AccessController.checkPermission(Permission portletPermission)
    4. 调用jetspeed自定义策略从保护域中检查是否拥有权限RdbmsPolicy.implies(ProtectionDomain protectionDomain, Permission permission
    5. 从数据库获取当前用户拥有的所有权限，检查这些权限是否隐含该portlet的访问权限
    6. 如果返回true，渲染portlet，如果返回false，没有权限，不渲染portlet

2) Jetspeed中标准用户属性的使用方法

用户标准属性在PortletRequest的内部类P3PUserInfos中作了明确定义

从PortletRequest中可以获取用户的标准属性，在portlet.xml中配置后即可生效

在portlet中获取用户标准属性的底层机制如下：

(1) 在portlet.xml配置user-attribute后，portlet应用启动时会将这些用户标准属性写到user-attribute表中，

(2) 引擎Jetspeed的类UserInfoManagerImpl，负责将用户的userPrincipal中的userInfoMap取出来，而userInfoMap实际就是该用户在security-attribute表中所有的自定义属性。

(3) 如果配置的用户属性在userInfoMap中存在，就取出结果，放在一个map中，反之，则不取。

(4) 返回存放结果的map

Portlet.xml配置如下：

<user-attribute>

<description>User Given Name</description>

<name>user.name.given</name>

</user-attribute>

Java代码调用如下：

Map attribute = (Map)request.getAttribute(RenderRequest.USER\_INFO);

String givenName = (userInfo!=null) ? (String)userInfo.get("user.name.given") : "";

用户属性也可以做映射，在Jetspeed-portlet.xml的配置如下：

<js:user-attribute-ref>

<js:name>user-name-given</js:name>

<js:name-link>user.name.given</js:name-link>

</js:user-attribute>

Java代码调用如下：

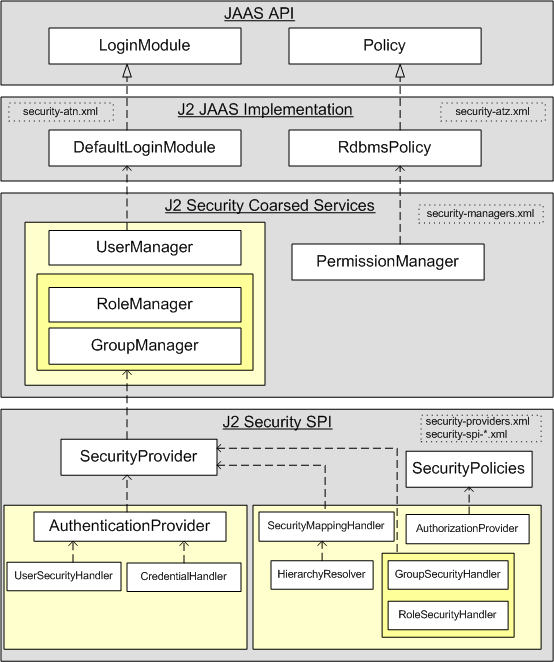
Map userInfo = (Map)request.getAttribute(PortletRequest.USER\_INFO);

String givenName = (userInfo!=null) ? (String)userInfo.get("user-name-given") : "";

2、关键技术、核心算法、思想

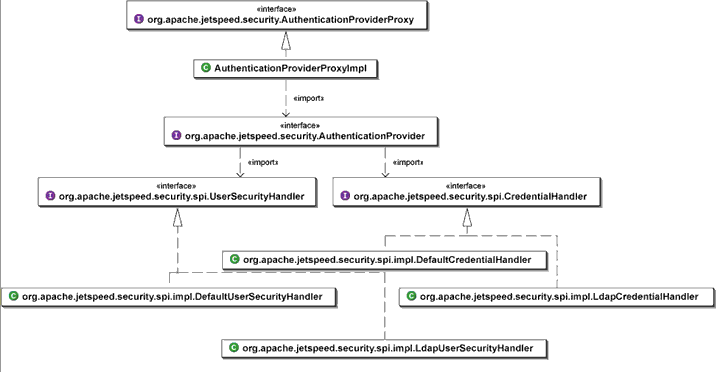
本模块是Jetspeed基于jaas实现的验证和权限模块

架构图如下：



3、类图

验证的核心类图：



授权的核心类图：

**RdbmsPolicy**：Jetspeed扩展Jaas的Policy，自定义的策略，它保证了用户权限来源于数据库。因为Jaas默认Policy的权限是保存在文件中的，而RdbmsPolicy覆盖了imply方法：通过调用PermissionManager，从数据库的security\_domain等表中获取权限permission。

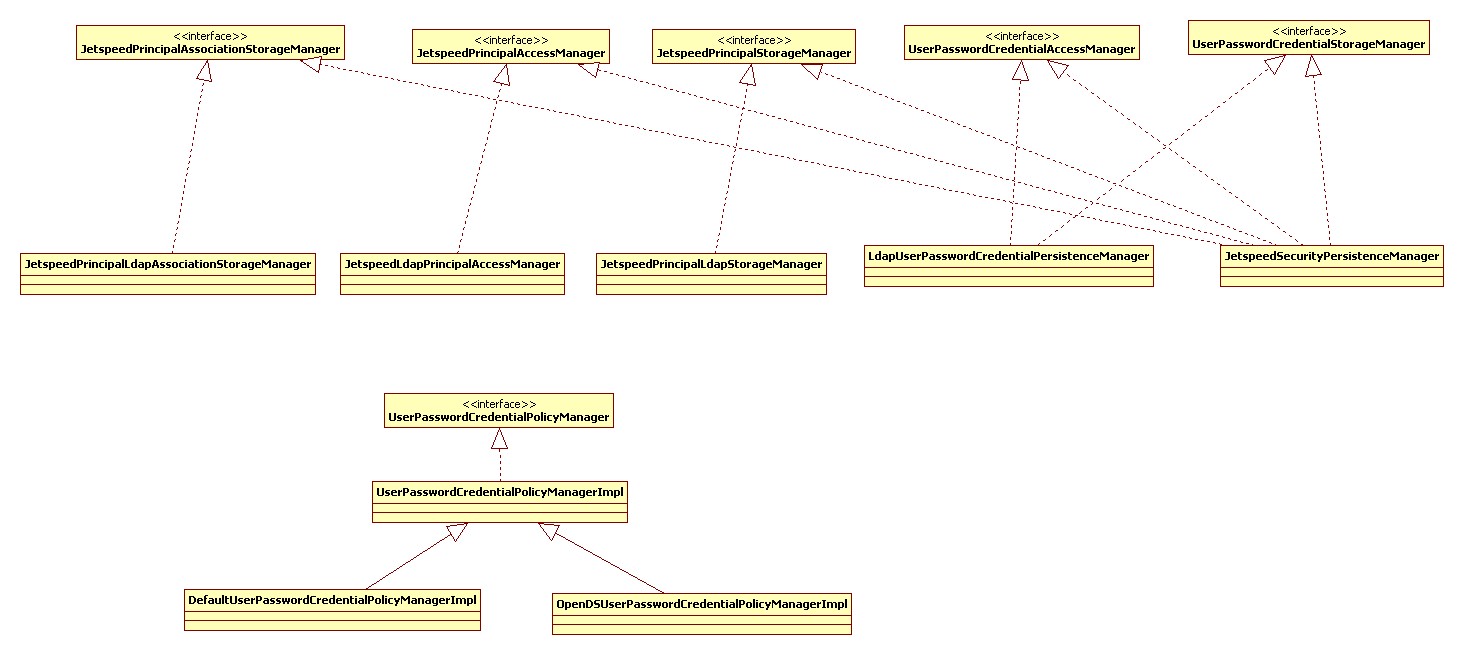
**PermissionManager**：获取与permission机制相关的权限信息。

1. 针对用户授予权限，废除权限
2. 新建某种类型的权限，可以是自定义的权限，也可以是默认权限类型page，folder，portlet和fragment

图一：



Ldap管理的核心类图

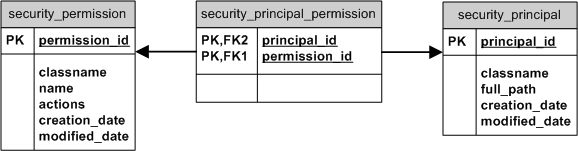


### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表



### 参考资料

1、wiki上的记录

2、官网链接页面

*http://portals.apache.org/jetspeed-2/devguide/guide-security.html*

### 问题记录

## jetspeed-sso

### 设计概述

1、模块依赖

该模块相对比较独立，是对jetspeed-api模块SSO管理相关接口的实现。

2、整体功能描述

本模块提供对单点登录站点、用户的增、删、改、查等管理，以实现通过IFrame方式来模拟单点登录。J2-admin工程间接调用本模块的具体实现，特别是通过SSOIFramePortlet这个portlet,，在编辑模式下可以新增、编辑站点。获取站点信息后，通过IFrame模拟进行登录对应站点

### 系统处理流程

本条描述模块的主要处理流程，

1. IFrame模拟登录序列



说明：当客户端请求包含SSOIFramePortlet的页面时，会请求到此portlet的doView()，从偏好中取出此portlet对应的站点URL，流程如上图。

### 设计思想

SSO模块主要的类关系图，核心的接口为SSOManagerImpl,可以通过此接口实现对SSOSite和SSOUser的管理

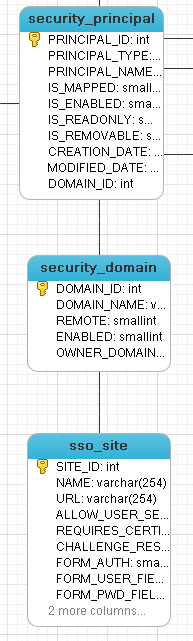


### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列



### 参考资料

1、wiki上的记录

2、官网链接页面

*http://portals.apache.org/jetspeed-2/deployguide/config-sso.html*

### 问题记录

## jetspeed-statistics

### 设计概述

1、模块依赖



2、整体功能描述

本模块主要分为两部分：

1、Audit 主要用于管理员，以及用户活动的记录统计。

* + - 1. 用户的相关活动。
      2. 管理员的相关活动。

2、Statistics主要用于记录用户的登入登出、页面以及Portlet的访问的记录统计。

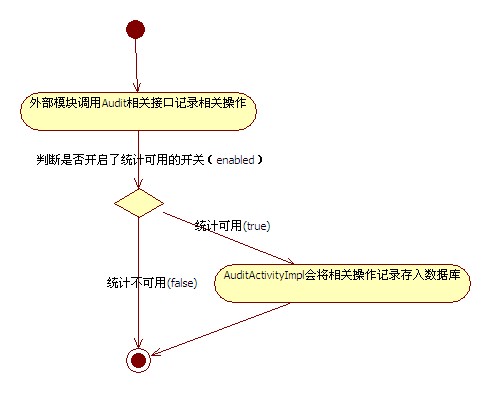
1. 在线用户的统计
2. Portlet的访问统计
3. 页面(page)的访问统计

### 系统处理流程

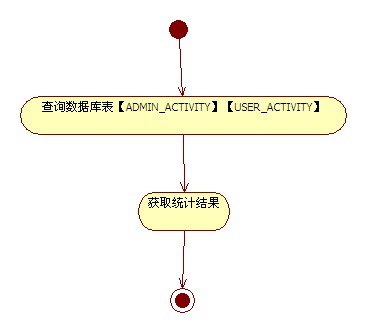
本条描述模块的主要处理流程，

1、 Audit处理流程

1) Audit记录流程

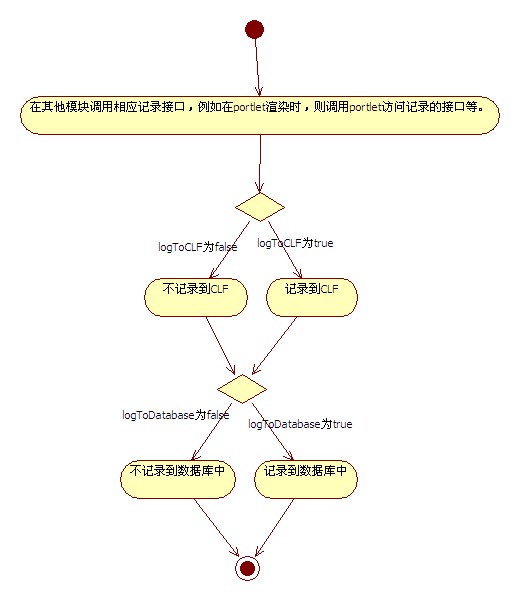


2) Audit查询流程

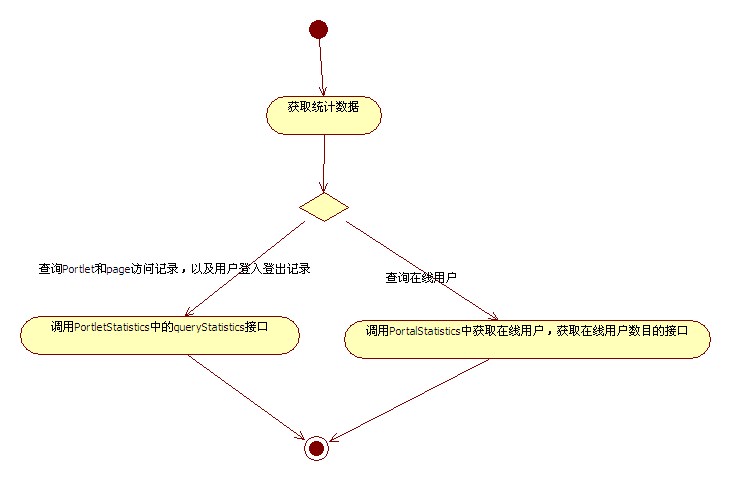


2、Statistics处理流程

1) Statistics记录流程



* + 1. Statistics查询流程



### 设计思想

1. 深入分析

在其他模块中，进行了一系列操作后，调用统计的接口，记录操作。同时，对于Audit的相关记录，接口提供了一个开关（即：Enabled），若enabled为true则记录到数据库中。而对于Statistics 则提供了logToCLF和logToDatabase两个开关，若两个都为false，则不记录。若logToCLF为true，则记录到CLF(Common Log Format file)。若logToDatabase为true，则记录到数据库中。

若需获取Audit的相关记录，则需要自行去查询数据库表【ADMIN\_ACTIVITY】【USER\_ACTIVITY】表。Jetspeed本身没有提供相应接口。

若需获取Statistics的相关记录，Jetspeed提供了接口查询在线用户，以及在线用户的数目。并提供了查询统计的接口queryStatistics（需要设置StatisticsQueryCriteria），并得到一个AggregateStatistics。根据AggregateStatistics获取所需数据。

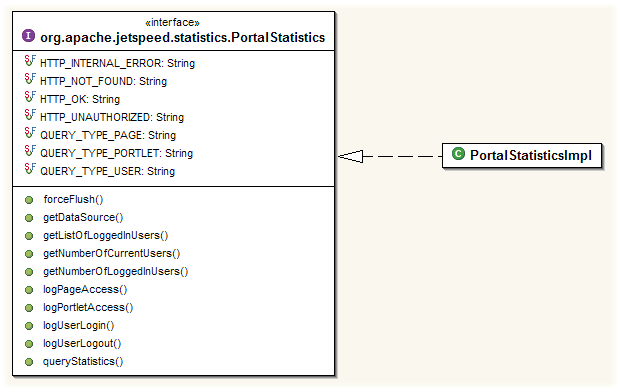
1. 关键技术、核心算法、思想

3、类图

1)Audit主要接口类。主要包含记录用户，管理员相关活动的接口。



2)Statistics主要接口类。主要包含记录用户登入登出、页面及Portlet访问的接口，以及在线用户查询，统计记录查询的接口。



### 重要数据模型

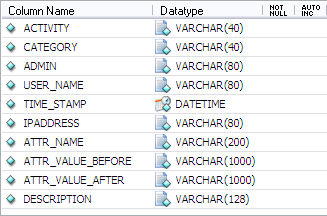
1、以数据结构图描述相互关系

2、数据结构设计。

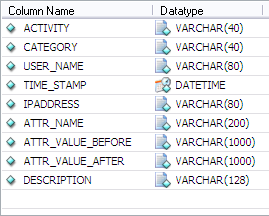
关联的表

1. Audit

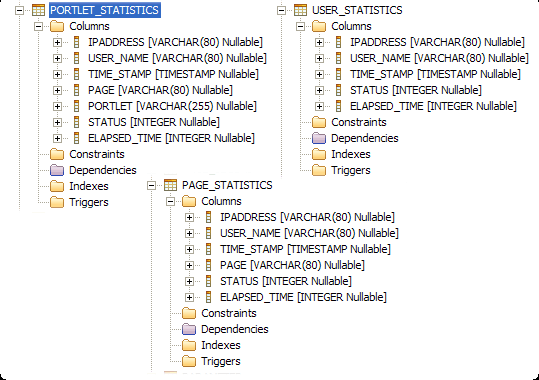
ADMIN\_ACTIVITY



USER\_ACTIVIYT



1. Statistics



### 参考资料

1、wiki上的记录

2、官网链接页面

<http://portals.apache.org/jetspeed-2/devguide/dev-statistics.html>

### 问题记录

## jetspeed-db-tools

### 设计概述

1、模块依赖



如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

例如portalsite模块，在POM里面没有和任何模块有依赖，但是portalsite从request里面获取了locator，为portal模块提供了menu。另外pagemanager模块也有和portalsite看起来类似的menuDefinition实现类，它们的关系也应该描述清楚。

2、关键技术、核心算法、思想

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-layout-portlets

### 设计概述

1、模块依赖

2、整体功能描述

本模块用于jetspeed的布局

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析

2、关键技术、核心算法、思想

本模块开发了jetspeed的默认布局portlet

LayoutPortlet和MultiColumnPortlet，但APS扩展了自己的布局Portlet---NestedLayoutPortlet，wiki上有很多设计相关的资料，这里就不细说了。

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-page-layout

### 设计概述

1、模块依赖



2、整体功能描述

本模块用于页面渲染时，重新包装页面、布局、portlet和权限的信息

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）

2、xx处理流程

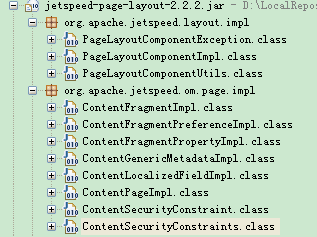
### 设计思想

1、深入分析

2、关键技术、核心算法、思想

当导航到一个页面时，需要根据最先导航状态（布局和普通portlet是什么模式和状态），重新包装新的页面内容，其中包括ContentPage，ContentFragment，ContentSecurityConstraint

如下图：



3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-capability（暂不处理）

### 设计概述

1. 模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

1. 整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1. xx处理流程
   1. 基于接口的时序图（或流程图）
   2. 数据流图、消息流图
2. xx处理流程

### 设计思想

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

1. 深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

1. 关键技术、核心算法、思想
2. 类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1. 以数据结构图描述相互关系
2. 数据结构设计。

capability

mimetype

media\_type

client

client\_to\_mimetype

mediatype\_to\_mimetype

### 参考资料

1. wiki上的记录
2. 官网链接页面

### 问题记录

## jetspeed-utils（暂不处理）

### 设计概述

1、模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

例如portalsite模块，在POM里面没有和任何模块有依赖，但是portalsite和locator的关系紧密。

2、关键技术、核心算法、思想

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-security-schema（暂不处理）

### 设计概述

1、模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

例如portalsite模块，在POM里面没有和任何模块有依赖，但是portalsite和locator的关系紧密。

2、关键技术、核心算法、思想

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-api（暂不处理）

### 设计概述

1、模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

例如portalsite模块，在POM里面没有和任何模块有依赖，但是portalsite和locator的关系紧密。

2、关键技术、核心算法、思想

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

## jetspeed-commons（暂不处理）

### 设计概述

1、模块依赖

使用pom依赖图表明依赖的模块，

如果是被别的模块所依赖，则用文字描述。

2、整体功能描述

整体描述

1. 功能点1
2. 功能点2

### 系统处理流程

本条描述模块的主要处理流程，

1、xx处理流程

* 1. 基于接口的时序图（或流程图）
  2. 数据流图、消息流图

2、xx处理流程

### 设计思想

1、深入分析（看情况完成）

描述清楚是如何和其他模块关联的，不单只是POM里面的依赖。

例如portalsite模块，在POM里面没有和任何模块有依赖，但是portalsite和locator的关系紧密。

2、关键技术、核心算法、思想

3、类图

（即使不画类图），并指出其中关键的类，及其作用

### 重要数据模型

1、以数据结构图描述相互关系

2、数据结构设计。

关联的表列出来

### 参考资料

1、wiki上的记录

2、官网链接页面

### 问题记录

*未研究的模块:*

序号模块优先级时间(负责人) 备注(数字越大优先级越高)

* + - 1. rdbms 1 一天（yxq）
      2. cm 4 二天（yxq）
      3. page-manager 缓存 5一天 (yxq)
      4. security 5 二天（yxq）
      5. file-cache 4 一天（yp）
      6. search 5 1.5天（yp）
      7. portal-site 2 二天（folderproxy 等proxy，及其缓存）(yp、sx)
      8. portlet-factory 4 一天(sx)
      9. registry 4 二天(sx)
      10. portal 5 3天本模块是其他模块的总入口，先研究好其他模块（yp）
      11. capability 1 目前没有多客户端支持需求，暂不研究
      12. header-resource 1（portlet中css和js重复加载的问题，可交由眀炜研究）
      13. sso 0 Jetspeed和APS实现自己的单点登录方式相似，暂不研究