DSpace Services Framework

This is not the DSpace 6 Services + Hibernate section

ⓘ This document refers to 1.6 to DSpace 2.0 code, i.e. dspace-services, this is very old. You are likely looking for the DSpace 6 refactoring that introduces clean business services and the Hibernate refactoring, look at: DSpace Service based api

DSpace Services Framework is maintained as a Supported Addon Module Dependency for DSpace 1.6 - 2.x

Subversion Access

http://scm.dspace.org/svn/repo/modules/dspace-services

Issue Management

http://jira.dspace.org/jira/browse/DSRV

Installation

Under development...

Documentation

- Under development...
  DSpace Services Framework
- The TAO of DSpace Services
- DSpace Spring Services Tutorial

Other Resources

- Mark Diggory's talk at DSUG09 on Backporting 2.0 Services
- DSpace 2.0 Modelling Services
- DSpace 2.0 Core Services

Implementing Providers

TODO: Provide examples of Implementing and Configuring Services in Spring and Guice

TODO: Provide examples of Implementing and Registering EventListeners in Spring and Guice.

Configuring Event Listeners

Event Listeners can be created by overriding the the EventListener interface:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans>

    <bean id="dspace" class="org.dspace.utils.DSpace"/>

    <bean id="dspace.eventService"
          factory-bean="dspace"
          factory-method="getEventService"/>

    <bean class="org.my.EventListener">
        <property name="eventService" >
                <ref bean="dspace.eventService"/>
          </property>
    </bean>
</beans>
```

TODO: Provide examples of Implementing and Registering Configurations in Spring and Guice.

Java Analogy – This Java code will perform the exact same steps as the above Spring configuration.

```java
//Initialize DSpace utility/helper object (to interact with the Kernel)
DSpace dspace = new DSpace();

//Get the EventService registered with the DSpace Kernel
EventService eventService = dspace.getEventService();

//Register a new EventListener with the existing EventService
//(Allows you to perform an action in your EventListener whenever an event is fired)
EventListener listener = new org.my.EventListener();
eventService.registerEventListener(listener);
```

Note: The above is completely valid Java code, assuming you create a new `org.my.EventListener` class, or replace it with a different, existing listener.

Architectural Overview

DSpace Kernel

The DSpace Kernel manages the start up and access services in the DSpace Services framework. It is meant to allow for a simple way to control the core parts of DSpace and allow for flexible ways to startup the kernel. For example, the kernel can be run inside a single webapp along with a frontend piece (like JSPUI) or it can be started as part of the servlet container so that multiple webapps can use a single kernel (this increases speed and efficiency). The kernel is also designed to happily allow multiple kernels to run in a single servlet container using identifier keys.

Kernel registration

The kernel will automatically register itself as an MBean in when it starts up so that it can be managed. It allows startup and shutdown and provides direct access to the ServiceManager and the ConfigurationService. All the other core services can be retrieved from the ServiceManager by their APIs.
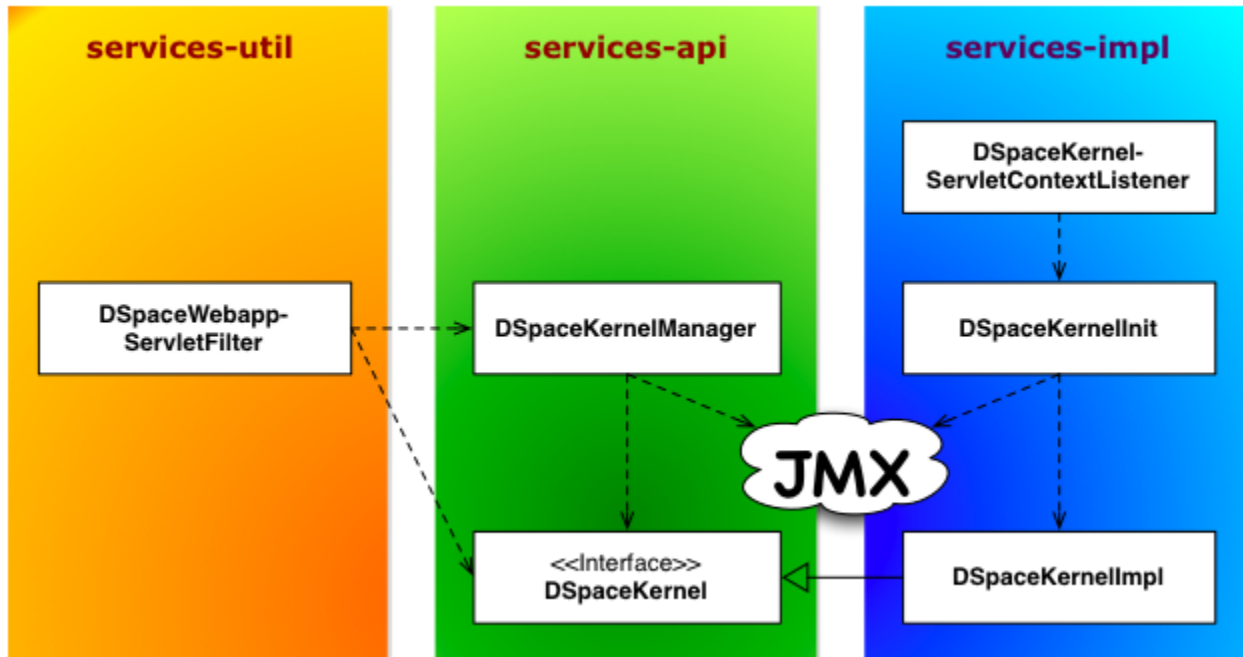
Kernel registration is currently handled in the following ways:

- XMLUI or JSPUI application – Load the `org.dspace.servicemanager.servlet.DSpaceKernelServletContextListener` in their respective `web.xml` files. This Listener initializes a new Kernel, which "lives" until the Servlet Context is destroyed.
- 'dspace' command-line launcher – The `org.dspace.app.laucher.ScriptLauncher` class initializes a new Kernel to run the command (and destroys it after the command is completed).

Kernel Startup and Access

The kernel can be started and accessed through the use of Servlet Filter/ContextListeners which are provided as part of the DSpace 2 utilities. Developers don't need to understand what is going on behind the scenes and can simply write their applications and package them as webapps and take advantage of the services which are offered by DSpace 2. Access to the kernel is provided via the Kernel Manager and the DSpace object which will locate the kernel object and allow it to be used.

```
/* Instantiate the Utility Class */
DSpace dspace = new DSpace();


/* Access get the Service Manager by convenience method */
ServiceManager manager = dspace.getServiceManager();


/* Or access by convenience method for default services */
EventService service = dspace.getEventService();
```
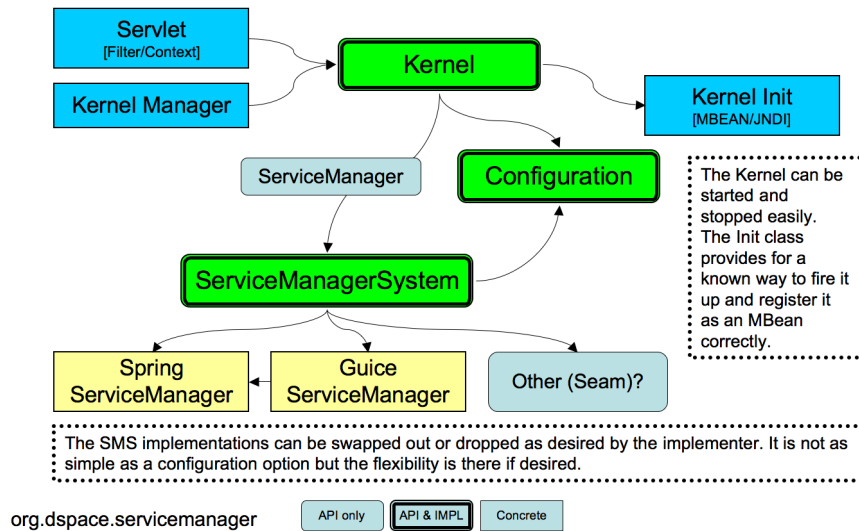
Service Manager

The ServiceManager abstracts the concepts of service lookups and lifecycle control. It also manages the configuration of services by allowing properties to be pushed into the services as they start up (mostly from the ConfigurationService). The ServiceManagerSystem abstraction allows the DSpace ServiceManager to use different systems to manage it's services. The current implementations include Spring and Guice. This allows DSpace 2 to have very little service management code but still be flexible and not tied to specific technology. Developers who are comfortable with those technologies can consume the services from a parent Spring ApplicationContext or a parent Guice Module. The abstraction also means that we can replace Spring/Guice or add other dependency injection systems later without requiring developers to change their code. The interface provides simple methods for looking up services by interface type for developers who do not want to have to use or learn a dependency injection system or are using one which is not currently supported.

DSpace 2 Service Manager
(DS2 Kernel / Service Manager)

Servlet [Filter/Context]	
Kernel Manager	

Kernel → **Kernel Init** [MBEAN/JNDI]

ServiceManager → **Configuration**

> The Kernel can be started and stopped easily. The Init class provides for a known way to fire it up and register it as an MBean correctly.

ServiceManagerSystem

Spring ServiceManager ← **Guice ServiceManager** Other (Seam)?

> The SMS implementations can be swapped out or dropped as desired by the implementer. It is not as simple as a configuration option but the flexibility is there if desired.

org.dspace.servicemanager

API only | API & IMPL | Concrete

The DS2 kernel is compact so it can be completely started up in a unit test (technically integration test) environment (this is who we test the kernel and core services currently). This allows developers to execute code against a fully functional kernel while developing and then deploy their code with high confidence.

Providers and Plugins

For developers (how we are trying to make your lives easier): The DS2 ServiceManager supports a plugin/provider system which is runtime hot-swappable. The implementor can register any service/provider bean or class with the DS2 kernel ServiceManager. The ServiceManager will manage the lifecycle of beans (if desired) and will instantiate and manage the lifecycle of any classes it is given. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the service if desired. The goal of this system is to allow DS2 to be extended without requiring any changes to the core codebase or a rebuild of the code code.

Activators

Developers can use an activator to allow the system to startup their service or provider. It is a simple interface with 2 methods which are called to startup the provider(s) and later to shut them down. These simply allow a developer to run some arbitrary code in order to create and register services if desired. It is the method provided to add plugins directly to the system via configuration as the activators are just listed in the configuration file and the system starts them up in the order it finds them.

Provider Stacks

Utilities are provided to assist with stacking and ordering providers. The priority is handled via a priority number such that 1 is the highest priority and something like 10 would be lower. 0 indicates that priority is not important for this service and can be used to ensure the provider is placed at or near the end without having to set some arbitrarily high number.

The DSpace Services Framework is a backporting of the DSpace 2.0 Development Groups work in creating a reasonable and abstractable "Core Services" layer for DSpace components to operate within. The Services Framework represents a "best practices" for new DSpace architecture and implementation of exentsions to the DSpace application. DSpace Services are best described as a "Simple Registry" where plugins. The DS2 (DSpace 2.0) core services are the main services that make up a DS2 system. These includes services for things like user and permissions management and storage and caching. These services can be used by any developer writing DS2 plugins (e.g. statistics), providers (e.g. Authn), or user interfaces (e.g. JSPUI).

Core Services

The core services are all behind APIs so that they can be reimplemented without affecting developers who are using the services. Most of the services have plugin/provider points so that customizations can be added into the system without touching the core services code. For example, let's say a deployer has a specialized authentication system and wants to manage the authentication calls which come into the system. The implementor can simply implement an AuthenticationProvider and then register it with the DS2 kernel ServiceManager. This can be done at any time and does not have to be done during Kernel startup. This allows providers to be swapped out at runtime without disrupting the DS2 service if desired. It can also speed up development by allowing quick hot redeploys of code during development.

Caching Service

Provides for a centralized way to handle caching in the system and thus a single point for configuration and control over all caches in the system. Provider and plugin developers are strongly encouraged to use this rather than implementing their own caching. The caching service has the concept of scopes so even storing data in maps or lists is discouraged unless there are good reasons to do so.

Configuration Service

The ConfigurationService controls the external and internal configuration of DSpace 2. It reads in properties files when the kernel starts up and merges them with any dynamic configuration data which is available from the services. The service allows settings to be updated as the system is running and also provides listeners which allow services to know when their configuration settings have changed and take action if desired. It is the central point to access and manage all the configuration settings in DSpace 2.

Manages the configuration of the DSpace 2 system. Can be used to manage configuration for providers and plugins also.

EventService

Handles events and provides access to listeners for consumption of events.

RequestService

In DS2 a request is the concept of a request (HTTP) or an atomic transaction in the system. It is likely to be an HTTP request in many cases but it does not have to be. This service provides the core services with a way to manage atomic transactions so that when a request comes in which requires mutliple things to happen they can either all suceed or all fail without each service attempting to manage this independently. In a nutshell this simply allows identification of the current request and the ability to discover if it succeeded or failed when it ends. Nothing in the system will enforce usage of the service but we encourage developers who are interacting with the system to make use of this service so they know if the request they are participating in with has succeeded or failed and take appropriate actions.

SessionService

In DS2 a session is like an HttpSession (and generally is actually one) so this service is here to allow developers to find information about the current session and to access information in it. The session identifies the current user (if authenticated) so it also serves as a way to track user sessions. Since we use HttpSession directly it is easy to mirror sessions across multiple servers in order to allow for no-interruption failover for users when servers go offline.