DSpace Service Manager

Introduction

The DSpace Spring Service Manager supports overriding configuration at many levels.

Configuration

Configuring Addons to Support Spring Services

Configuring Addons to support Spring happens at two levels. Default Spring configuration is available in the DSpace JAR or WAR resources directory and allows the addon developer to inject configuration into the service manager at load time. The second level is in the deployed [dspace]/config/spring directory where configurations can be provided on a addon module by addon module basis.

This latter method requires the addon to implement a SpringLoader to identify the location to look for Spring configuration and a place configuration files into that location. This can be seen inside the current [dspace-source]/config/modules/spring.cfg

Configuration Priorities

The ordering of the loading of Spring configuration is the following:

1. configPath = "spring/spring-dspace-applicationContext.xml" relative to the current classpath
2. addonResourcePath = "classpath*:spring/spring-dspace-addon-*-services.xml" relative to the current classpath
3. coreResourcePath = "classpath*:spring/spring-dspace-core-services.xml" relative to the current classpath
4. Finally, an array of SpringLoader API implementations that are checked to verify "config/spring/module" can actually be loaded by its existence on the classpath. The configuration of these SpringLoader API classes can be found in dspace.dir/config/modules/spring.cfg.

Configuring a new Addon

There are 2 ways to create a new Spring addon: a new Spring file can be located in the resources directory or in the configuration [dspace]/config/spring directory. A Spring file can also be located in both of these locations but the configuration directory gets preference and will override any configurations located in the resources directory.

Addon located as resource in jar

In the resources directory of a certain module, a Spring file can be added if it matches the following pattern: "spring/spring-dspace-addon-*-services.xml". An example of this can be found in the dspace-discovery-solr block in the DSpace trunk. (`spring-dspace-addon-discovery-services.xml`) Wherever this jar is loaded in a Maven module, the Spring files will be processed into services.

Addon located in the [dspace]/config/spring directory

This directory has the following subdirectories in which Spring files can be placed:

- api: when placed in this module the Spring files will always be processed into services (since all of the DSpace modules are dependent on the API).
- discovery: when placed in this module the Spring files will only be processed when the discovery library is present

The reason why there is a separate directory is that if a service cannot be loaded, the kernel will crash and DSpace will not start.

Configuring an additional subdirectory for a custom module

So you need to indeed create a new directory in [dspace]/config/spring. Next you need to create a class that inherits from the "org.dspace.kernel.config. SpringLoader". This class only contains one method named getResourcePaths(). What we do now at the moment is implement this in the following manner:

```
@Override
public String[] getResourcePaths(ConfigurationService configurationService) {
     StringBuffer filePath = new StringBuffer();
     filePath.append(configurationService.getProperty("dspace.dir"));
     filePath.append(File.separator);
     filePath.append("config");
     filePath.append(File.separator);
     filePath.append("spring");
     filePath.append(File.separator);
     filePath.append("{module.name}"); //Fill in the module name in this string
     filePath.append(File.separator);
     try {

          //By adding the XML_SUFFIX here it doesn't matter if there should be some kind of spring.xml.old file
in there it will only load in the active ones.
          return new String[]{new File(filePath.toString()).toURI().toURL().toString() + XML_SUFFIX};
      } catch (MalformedURLException e) {
          return new String[0];
      }
}
```

After the class has been created you will also need to add it to the "spring.springloader.modules" property located in the [dspace]/config/modules/spring.cfg.
The Spring service manager will check this property to ensure that only the interface implementations which it can find the class for are loaded in.

By doing this way we give some flexibility to the developers so that they can always create their own Spring modules and then Spring will not crash when it can't find a certain class.

The Core Spring Configuration

Utilizing Autowiring to minimize configuration complexity.

Please see the following tutorials:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

Accessing the Services Via Service Locator / Java Code

Please see the following tutorials:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services

Architectural Overview

Please see Architectural Overview here: DSpace Services Framework

Service Manager Startup in Webapplications and CLI

Please see the DSpace Services Framework

Tutorials

Several good Spring / DSpace Services Tutorials are already available:

- DSpace Spring Services Tutorial
- The TAO of DSpace Services