DSpace service based api: Porting DSpace modules

Build and deployment instructions for services-api (development)

Postgres changes

Enable pgcrypto extension

In order to generate UUID identifiers in postgres an extension module is required.

- n_random_uuid()" method, see Unable to locate Jira server for this macro. It may be due to Application Link configuration.)
  he "postgresql-contrib" package.
- Connect to the database using user postgres
- Run the following command: "CREATE EXTENSION pgcrypto;"
- The output of the command if successful is blank, so you can always try to run "SELECT gen_random_uuid();" which should return a result.

Database migration to service based api

Database migration occurs on tomcat start (handled by Flyway). Both existing databases & new installation should work (though please report any errors if you find any).

Maven build instructions

The same build instructions apply as in DSpace 5:

```
mvn clean package
```

Ant update

Don't forget to ant update/fresh_install before starting up tomcat again.

The "ant update" or "ant fresh_install" process will trigger a database update. If your database content is important to you (or you wish to test the migration process more thoroughly), you may wish to back it up prior to running Ant. The migration will obviously modify your existing database. While we think it is "safe", it is not considered "production ready" until 6.0 is released.

Starting tomcat

Finally, start up Tomcat as normal

How to migrate DSpace modules to the service api

Tutorial: Basic class porting

Before porting it is recommended to go over: DSpace Service based api again to make sure you understand the impact.

For this tutorial I will explain how one would port the HandleServlet found in the JSPUI module. This is a fairly simple class with a couple of services.

Step 1: Find & declare services

The first step when porting is to find all the services and declare them at the top. So scroll through the class & look for [serviceName]Impl.anyMethod missing methods on what are now database entities, below are two examples:

```
//Static call to a service method that will fail on compilation
AuthorizeServiceImpl.authorizeAction(context, item, Constants.READ);
//Another static call to a service method
HandleServiceImpl.resolveToObject(context, handle);
//Method not found on item since item.canEdit() is a business method call & not a simple getter/setter
item.canEdit();
//Method not found on collection since collection.canEditBoolean(true) is a business method call & not a simple getter/setter
collection.canEditBoolean(true);
```

For each of the services you need use a factory call to retrieve them (or if you are working in a spring bean you can autowire them).

In order to quickly find the factory call you need, see: DSpace Service based api: API Changelist, every service that has a factory call has an example of how to retrieve. The factories always have the following format: [packageName]ServiceFactory.getInstance().get[className]Service(). So after a while you won't even need to look them up.

When we add the factory calls at the top of the class (at the top declaration isn't mandatory, but best that we stick to a single way of how to handle things, this makes it easier to see which services are already declared) we get the following:

```
protected AuthorizeService authorizeService = AuthorizeServiceFactory.getInstance().getAuthorizeService();
protected HandleService handleService = HandleServiceFactory.getInstance().getHandleService();
protected CommunityService communityService = ContentServiceFactory.getInstance().getCommunityService();
protected CollectionService collectionService = ContentServiceFactory.getInstance().getCollectionService();
protected ItemService itemService = ContentServiceFactory.getInstance().getItemService();
protected SubscribeService subscribeService = EPersonServiceFactory.getInstance().getSubscribeService();
```

Always use the factories, never create an implementation by using its constructor. It will resolve compile issues but will crash when testing.

Step 2: Integrate the services

The next step is to use the factories to resolve a lot of compilation issues, below are some examples:

```
//Before
AuthorizeServiceImpl.authorizeAction(context, item, Constants.READ);
///After
authorizeService.authorizeAction(context, item, Constants.READ);

//Before
HandleServiceImpl.resolveToObject(context, handle);
//After
handleService.resolveToObject(context, handle);

//Before
item.canEdit();
//After (requires the item & a context, because the service is a singleton & doesn't hold a state the database
object should always be passed)
itemService.canEdit(context, item);

//Before
collection.canEditBoolean(true);
//After (requires the collection & a context, because the service is a singleton & doesn't hold a state the
database object should always be passed)
collectionService.canEditBoolean(context, collection, true);

//There could even be some other methods that now require an additional argument that didn't need it before:
//Before
xHTMLHeadCrosswalk.disseminateList(item);
//After (requires a context)
xHTMLHeadCrosswalk.disseminateList(context, item);
```

Step 3: Fix the rest

As seen in the example above some of the services require additional arguments, depending on the method this could be a context, database object, ... A complete list of methods that have changed in this manner is available here DSpace Service based api: API Changelist

Another change easily encountered is the fact that in older DSpace versions an array of objects was returned instead of a list. This has been changed to always return lists. So the following changes will also need to be made:

```
//Before
Community[] parents = c.getCommunities();
request.setAttribute("dspace.community", parents[0])
//After
List<Community> parents = c.getCommunities();
request.setAttribute("dspace.community", parents.get(0))
```

Use the shortcuts of your idea to quickly traverse these issues

Things to out for when porting

Below are a few additional things to look out for when porting, these have not yet been mentioned in: DSpace Service based api#ChangestotheDSpaceapi

- DatabaseManager calls are no longer allowed from the UI, if you find a call to the DatabaseManager, see if a similar method already exists, if not create add a new service method which delegates it to the DAO layer. There should be examples enough available.
- All lists returned from database objects are persistent, if you want to remove an item from this list you can't just use list.remove() since it will give you an exception when testing. Use an iterator & remove it from the iterator