Hooks

Hooks

This topic is discussed in the Advanced Topics training video.

Hooks are points in the Grouper API that will callback custom code if configured to do so. The custom code has the opportunity to be notified that something happened, change data as it is being operated on, veto operations, kickoff other logic (log something, call another grouper API method, etc).

Hooks are available for the following database persistable objects: Group, Stem, Member, Membership, Composite, Field, GrouperSession, GroupType, GroupTypeTuple

- Built in Grouper hook for one membership in folder
- Do not allow non sysadmins to add EveryEntity to groups or privileges
- Do not allow someone to add themself to a group
- Getting started with hooks
- Getting started with hooks2
- Grouper hook to ensure group does not exist in LDAP on group create
- Grouper hook to make sure same group extension is not used in folder(s)
- Grouper hook which adds link to UI
- Hooks POC (Proof of concept)

Each of these objects has low-level hooks associated with it: preInsert, postInsert, postCommitInsert, preUpdate, postUpdate, postCommitUpdate, preDelete, postDelete, postCommitDelete.

Note, some of these operations don't really make sense but are there anyway (e.g. there is no way to update a GroupType, and no way to delete Member).

The low-level hooks are called with each hibernate call for an object that generally corresponds to a row in a DB table (not for groups/attributes which have one hook but which are a one-to-many).

The pre is before the sql statement is sent to the DB, and the post is after, though both are before the SQL commit. The postCommit kicks off right after the commit is called in the database transaction (note: since we have long runing transactions this could be after a long workflow). You would use the pre to edit the object or to insert SQL before the sql. The post is used to get the hibernate id of an insert (e.g. to add rows with foreign keys). Both can be used to veto a call if synchronous [default]. Asynchronous hooks, and postCommit hooks cannot veto. Post commit hooks do not participate in the same database transaction as the business logic. Post commit hooks can be used for economical notifications (note, there are more robust strategies also). Post commit and asynchronous hooks receive copies (clones) of the business objects so the data is consistent and safe.

There are high-level hooks which encompass multiple low-level operations, e.g. addMember, removeMember (only 2 so far)

Finally there are hooks on general lifecycle activities in Grouper, called LifecycleHooks. Examples are: when grouper starts up, when hooks start (to register suites of hooks), and when hibernate is being configured (to register another hibernate mapping into the grouper hibernate configuration session factory).

Configure

To configure a hook, subclass one of the hooks base classes, e.g. GroupHooks, MembershipHooks, etc. Then either register this in the grouper.properties (see grouper.example.properties[broken] for documentation). Note the class must be threadsafe, as an instance is cached and called repeatedly (i.e. dont store instance vars etc)

```
#implement a group hook by extending edu.internet2.middleware.grouper.hooks.GroupHooks
hooks.group.class=edu.yourSchool.it.YourSchoolGroupHooks,edu.yourSchool.it.YourSchoolGroupHooks2
```

Note you can register multiple hooks by comma separating them. Or you can register runtime with:

```
GrouperHookType.addHookManual("hooks.group.class", YourSchoolGroupHooks2.class);
```

If you want to register a suite of hooks, just register a lifecycle hook:

```
hooks.lifecycle.class=edu.internet2.middleware.grouper.hooks.LifecycleHooksImpl
```

Then in the hook init method, register some hooks for the suite:

```
package edu.internet2.middleware.grouper.hooks;

import edu.internet2.middleware.grouper.hooks.beans.HooksContext;
import edu.internet2.middleware.grouper.hooks.beans.HooksLifecycleHooksInitBean;
import edu.internet2.middleware.grouper.hooks.logic.GrouperHooksUtils;


/**
 *
 */
public class LifecycleHooksImpl extends LifecycleHooks {

  /**
   * @see edu.internet2.middleware.grouper.hooks.LifecycleHooks#hooksInit(edu.internet2.middleware.grouper.
hooks.beans.HooksContext,
   *            edu.internet2.middleware.grouper.hooks.beans.HooksLifecycleHooksInitBean)
   */
  @Override
  public void hooksInit(HooksContext hooksContext,
      HooksLifecycleHooksInitBean hooLifecycleHooksInitBean) {
    GrouperHooksUtils.addHookManual("hooks.group.class", MyGroupHooks.class);
    GrouperHooksUtils.addHookManual("hooks.stem.class", MyStemHooks.class);
    GrouperHooksUtils.addHookManual("hooks.member.class", MyMemberHooks.class);
  }

}
```

Finally, I picture built-in Grouper suites to just insert the registration based on Grouper config property in GrouperHooksUtils in the initHooks block. To enable the suite the grouper properties file would just have a Boolean switch. When someone wants to get this working, let me know the Grouper properties config property and the LifecycleHooks subclass and I can do an example.

API

Each hook method gets two arguments as callbacks. The hooks context, and the bean that holds the relevant data for the hook.

The hooks context (still needs to be completed), holds information such as the current user, the current environment (UI vs WS etc), threadlocal data (e.g. the current http request), etc. The hooks bean holds information such as the Group which is being deleted. If any new arguments need to be added to a hook method, they will be added to one of these beans, so the signature of the method wont change, so people dont have to recompile on upgrade.

To find out how the data has changed (in an update or delete), you can use the dbVersion API. Note, this is probably only available in the "pre" hooks.

To veto a hook, throw a HookVeto which is a runtime exception. The specific hooks which is vetoing will be assigned to the exception, and will be known wherever it is caught. This HookVeto takes a system name and a friendly description of a reason why it is being vetoed. The system name can be used to look up a localized error message. The friendly version can be used for logging or if a localized message doesnt exist. You can only veto pre and post synchronous hooks. You cannot veto asynchronous or postCommit hooks. You also cannot veto lifecycle hooks (e.g. hibernate init).

Hooks rely on all DB code going through the Grouper hibernate API (see the class HibernateSession). If any DB code uses the Session object (from hibernate) directly, then hooks will not fire. You can use the Grouper Hibernate API and circumvent hooks like this (e.g. in a reset() method):

```
        hibernateSession.byObject.setIgnoreHooks(true).delete(_type.getFields());
```

If you want to make a hook non-reentrant (while in hook, do not let that particular hook fire), you can easily setup a threadlocal to accommodate. e.g.

```
private static ThreadLocal<Boolean> inOnPreUpdate = new ThreadLocal<Boolean>();

  /**
   * @see edu.internet2.middleware.grouper.hooks.StemHooks#stemPostUpdate()
   */
  @Override
  public void stemPreUpdate(HooksContext hooksContext, HooksStemBean postUpdateBean) {

    Boolean inOnPreUpdateBoolean = inOnPreUpdate.get();
    try {

      if (inOnPreUpdateBoolean == null || !inOnPreUpdateBoolean) {
          inOnPreUpdate.set(true);          ... do logic ...
        }
      } finally {
        //if we changed it
        if (inOnPreUpdateBoolean== null || !inOnPreUpdateBoolean) {
          //change it back
          inOnPreUpdate.remove();
        }
      }


  }
```

The HooksContext contains a map of attributes. These attributes can be set with static methods in HooksContext, and each attribute is designated as allowed to be copied to another thread or not. e.g. the HttpServletRequest is only valid during a request, so it shouldnt be available in another thread which might last longer than the request. There are some constants in HooksContext for common keys. Some of the built-in values for the attributes are: HttpServletRequest, HttpServletResponse, HttpSession (Http classes for UI and WS only). The hooks context also holds the current context name. This is retrieved with hooksContext.getGrouperContextType(). This can be assigned (with static method in GrouperContextTypeBuiltIn) in the current thread or global default. This is available everywhere in grouper, not just hooks. If it is not set, it is UNKNOWN. Also the current user is available from the context. There is the logged in user, the actAs user (perhaps WS only), and the GrouperSession user. You can make decisions based on which user is using the app. There are also convenience methods (e.g. boolean hooksContext.isSubjectFromGrouperSessionInGroup(groupName))

Asynchronous hooks

Hooks by default are synchronous: they run in the same thread as the thread which is doing the work that is hooked.

If you want the login of a hook to be asynchronous (e.g. to not slow down the current thread), then it will not be in the same transaction or have the same data available (e.g. not the HttpServletRequest in a web related hook). You cannot veto an asynchronous hook, or participate in the same database transaction as the business logic. Asynchronous hooks retrieve a different HooksContext (threadsafe), and a copy of the hooks bean (so no toes are stepped on, and the data is a snapshot). To do asynchronous hooks, there are two ways:

1. Make the hook implementation class implement the marker interface HookAsynchronousMarker (note all hook methods in this hook implementation will be asynchronous), e.g.

```
public class MembershipHooksImplAsync2 extends MembershipHooks implements HookAsynchronousMarker {
```

-or- 2. Call the asynchronous callback, anything in the callback is in a different thread [asynchronous]

```
 public class MembershipHooksImplAsync extends MembershipHooks {

  /**
   *
   */
  @Override
  public void membershipPreAddMember(HooksContext hooksContext,
      HooksMembershipChangeBean preAddMemberBean) {
        //do logic in same thread/transaction
    //also you can get data from the beans above, set final, and use below (if it wont be available otherwise,
this might be rare)
    HookAsynchronous.callbackAsynchronous(hooksContext, preAddMemberBean, new HookAsynchronousHandler() {

      public void callback(HooksContext hooksContext, HooksBean hooksBean) {

        HooksMembershipChangeBean preAddMemberBeanThread = (HooksMembershipChangeBean)hooksBean;
                //do logic in a different thread/transaction

      }

    });

  }

}
```

Logging

There is logging about which hooks execute when, for how long, and how they end (normal, veto, exception). To enable logging, add something like this to the log4j.properties:

```
# see hook debug info
log4j.logger.edu.internet2.middleware.grouper.hooks = DEBUG, grouper_debug
```

Then in the debug log, you will see info about all hooks (note, the ID's are for the purpose of matching the log statements start and end, also can be accessed from the HooksContext:

```
2008/07/09 02:18:05.482 GrouperHooksUtils.executeHook(174) - START: Hook GroupTypeTupleHooksImpl.
groupTypeTuplePreInsert id: PSPTRJ8J
2008/07/09 02:18:05.497 GrouperHooksUtils.executeHook(181) - END (normal): Hook GroupTypeTupleHooksImpl.
groupTypeTuplePreInsert id: PSPTRJ8J (15ms)
2008/07/09 02:18:05.497 GrouperHooksUtils.executeHook(174) - START: Hook GroupTypeTupleHooksImpl.
groupTypeTuplePostInsert id: PSPTRJ8K
2008/07/09 02:18:05.497 GrouperHooksUtils.executeHook(186) - END (veto): Hook GroupTypeTupleHooksImpl.
groupTypeTuplePostInsert id: PSPTRJ8K (0ms),
   veto key: hook.veto.groupTypeTuple.insert.name.not.test4, veto message: name cannot be test4
```

Use cases

- Validate that a group extension conforms to standards, veto if not
- Only grouperLoader itself (or someone in the wheel group), can add a member to a group with type "grouperLoader"
- A new or updated group emailAddress attribute should not be in use by another group or subject

To do

- Add more high level hooks
- Collect and code use cases

See also

Grouper Rules

Access Management Features Overview