Exception handling in Grouper

Summary

In general Grouper uses unchecked exceptions in grouper 1.5. This means the exception extends RuntimeException, and does not need an explicit try /catch.

- Easier for nested method called not to have to rethrow or repackage the exception into another
- Easier for inverse of control and other anonymous inner classes

Checked exceptions should be used for things which are expected to be caught and handled in most cases. Also, exceptions should not be used for simple flow control, but rather for when exception cases happen.

Finders

One complicated part of moving from checked to unchecked exceptions in grouper 1.5 was finders which return a single object. e.g.

- Group GroupFinder.findByUuid(uuid) throws GroupNotFoundException
- Group GroupFinder.findByName(name) throws GroupNotFoundException

In the findByUuid case, generally speaking if the group is not found, it is an error. In the findByName case, it might not be assumed that the group exists, and the caller is just checking to see if it exists, so when not found, it shouldnt really be an exception, it should return null. Also, there is confusion since the finders which return multiple results (a Set of Groups), return an empty set or null, and dont throw GroupNotFoundException when not found.

So in order to turn the GroupNotFoundException to runtime, the finders which return a single value need to change. This is because since not checked, it will not be obvious to the caller that they need to catch an exception if not found, they might assume it just returns null (like the multiple case). So instead of changing these methods to return null, they are now deprecated, and there is a sibling method which takes a boolean also which indicates if the object is expected to be found (exceptionIfNull).

Examples

Here is an example of exception handling in 1.4 which is no longer necessary. Im not sure if it is ideal to take this out or leave it in, not sure what the use is. But at least we will not have that choice

```
catch (InsufficientPrivilegeException eIP)  {
  throw new MemberAddException(eIP.getMessage() + ", " + errorString, eIP);
}
catch (MemberNotFoundException eMNF)        {
  throw new MemberAddException( eMNF.getMessage() + ", " + errorString, eMNF );
}
```

Here is another example which might not even be correct... are all schema exceptions the same meaning as AttributeNotFoundException???

```
catch (SchemaException eS) {
  throw new AttributeNotFoundException(eS.getMessage(), eS);
}
```

Here is an example of tunneling checked exceptions through an anonymous inner class (inverse of control). This is not needed anymore

```
              //wrap checked exceptions inside unchecked, and rethrow outside
            } catch (StemNotFoundException snfe) {
              throw new RuntimeException(snfe);
            } catch (InsufficientPrivilegeException ipe) {
              throw new RuntimeException(ipe);
            } catch (StemAddException sae) {
              throw new RuntimeException(sae);
            } catch (GroupAddException gae) {
              throw new RuntimeException(gae);
            }
          }

        });

    }
  });
  return group;
} catch (RuntimeException re) {

  Throwable throwable = re.getCause();
  if (throwable instanceof StemNotFoundException) {
    throw (StemNotFoundException)throwable;
  }
  if (throwable instanceof InsufficientPrivilegeException) {
    throw (InsufficientPrivilegeException)throwable;
  }
  if (throwable instanceof StemAddException) {
    throw (StemAddException)throwable;
  }
  if (throwable instanceof GroupModifyException) {
    throw (GroupModifyException)throwable;
  }
  if (throwable instanceof GroupNotFoundException) {
    throw (GroupNotFoundException)throwable;
  }
  if (throwable instanceof GroupAddException) {
    throw (GroupAddException)throwable;
  }
  //must just be runtime
  throw re;
```

Upgrade steps

- If you have custom code using the Grouper API, see if you are using an deprecated methods (eclipse should give you a warning), and if so, then use the sibling method which is not deprecated. Generally it is the same method with "true" as the last param, indicating that you expect exceptioned to be thrown if not found. This is the behavior of the deprecated method
- If you want to go the extra step, you could clean up your own exception handling based on Grouper having runtime exceptions... i.e. you wont need as much try/catching.

Proposal

Note: there were several emails which discuss this approach

Grouper 1.4- has a lot of checked exceptions. This is inconvenient, and is not the direction that Java seems to be going (e.g. hibernate v3, spring, etc). Most of these exceptions cannot be handled by callers, and end up being rethrown or converted into RuntimeExceptions. We should continue to try to document which runtime exceptions can be thrown from methods, though this is not a requirement and shouldnt be relied on.

I propose that we change the superclass of the core grouper exception to not be Exception, but instead RuntimeException (or if there is already an equivalent runtime exception, they should be merged).

the exceptions I propose changing are:

- GrantPrivilegeException
- GroupAddException
- GroupDeleteException
- GrouperException
- GrouperShellException
- GroupModifyException
- InsufficientPrivilegeException

- LdappcException
- MemberAddException
- MemberDeleteException
- QueryException
- RevokePrivilegeException
- SchemaException
- SessionException
- StemAddException
- StemDeleteException
- StemModifyException

The only exceptions which will remain checked will be the NotFound exceptions, though those methods should be overloaded so you can pick if you want the exception or null returned. If you want null returned, it should not throw the checked exception. We should look at the subject API to see what we should do.

The problem is the NotFound exceptions is that sometime it is a problem if it is not found (e.g. when looking up a foreign key uuid), but sometimes it is not a problem (e.g. seeing if a group exists by name). So lets make these runtime, and pass in a boolean is to if an exception is expected or not.