Infrastructure - Task #706

Story # 1079 (Closed): Decide how to handle the InputStreamFromOutputStream problem where no exceptions are passed out of D1Client

Exceptions not correctly passed through CRUD interface

2010-07-06 22:05 - Chad Berkley

Status:	Closed	**Start date:**	
Priority:	High	**Due date:**	
Assignee:	Chad Berkley	**% Done:**	100%
Category:	Metacat	**Estimated time:**	0.00 hour
Target version:	Sprint-2010.48		
Milestone:		**Story Points:**	
Product Version:	*		

Description

Metacat is currently using the InputStreamFromOutputStream class to read a stream from Metacat and pass it through the CRUD interface without writing it to memory first. This seems to work well except that exceptions are not passed through. Instead, InputStreamFromOutputStream seems to only be able to throw IOExceptions. No matter which exception is thrown in the produce() method (which is executed in a 2nd thread), an IOException is always thrown to the caller. This happens for any CRUD method that uses this methodology (currently get() and getSystemMetadata()).

This causes the client to get an erroneous exception of the form "Exception producing data" instead of the DataONE exception that was actually thrown.

We should possibly look into using memory mapped IO or some other method for handling large data streams instead of the InputStreamFromOutputStream methodology. I looked at their code and confirmed that they are trapping all exceptions and reporting them as an IOException. We could also contribute code to that project to fix this problem.

Until we figure out a way to handle this, I am adding additional logging information to metacat to point out this behavior in the logs. So if you get an exception about producing data, look at the metacat logs for more information.

History

#1 - 2010-07-06 22:19 - Matthew Jones

For an example of how to re-throw the proper exception, see the following blog post:
http://www.attivio.com/blog/56-java-development/287-i-have-an-output-stream-but-i-need-an-input-stream.html
I'm not sure how well their algorithm works, but it might be a replacement for the InputStreamFromOutoutStream class, or maybe we could add this type of an approach onto that class and contribute it back to the easystream project.

#2 - 2010-07-06 23:02 - Chad Berkley

Looks like to use the fix described in the link in the comment, we'd have to return InputStreamFromOutputStream from CrudService.get() instead of an InputStream.

#3 - 2010-07-08 16:07 - Chad Berkley

I tried recasting the InputStream returned by get as an InputStremToOutputStream then calling getResult(). I had limited success. It basically allows you to print any exception message, but it doesn't actually give you the exception that was thrown.

#4 - 2010-08-02 21:39 - Chad Berkley

See the TODO in CrudService.java

#5 - 2010-10-06 16:42 - Chad Berkley

- Parent task set to #912

#6 - 2010-11-15 19:53 - Chad Berkley

After looking at this further, there is really no good way to fix this. The outputstream must be written to in a separate thread. The only way we could potentially capture these exceptions would be to return an InputStreamFromOutputStream object instead of an InputStream. This seems to be the only way to capture any type of exceptions thrown from within the 2nd thread. This would make the client call be a bit strange, however. Instead of dealing with a normal InputStream, the user would need the jar for the InputStreamFromOutputStream and would have to handle it properly. I'm going to add this task to a new story and push it back into the backlog for now.

#7 - 2010-11-15 19:54 - Chad Berkley

- Parent task changed from #912 to #1079

#8 - 2010-12-02 00:09 - Chad Berkley

- Status changed from New to Closed

- % Done changed from 0 to 100

All of the produce methods have been converted to use file IO instead of the multithreaded approach. This allows the exceptions to be passed correctly. The OutputStream is written to a temp file, then the InputStream is read from the file. Once the read is complete, the file is deleted.