TikaBatchOverview

Documentation for the tika-batch module

This module is currently available in trunk and will be available in Tika 1.8. The code has been under development for a while, but there may be surprises. Please test early and often and submit issues when you find them. tika-batch is available as a standalone module, and it is integrated into tika-app.

The Need

William Palmer documents what many integrators of Tika face – Tika works very well on most documents, but it can run into problems. As Nick Burch notes on slides 47-55, even if Tika fails catastrophically on a small percentage of documents, a small percentage of a lot of documents is still a lot of documents, and "you need to plan for failures". Some types of catastrophic failures include:

- Permanent hangs (runaway parsers)
- Out-of-Memory Errors
- Memory leaks

Running Tika efficiently and making it robust against these problems are non-trivial issues, and it will be helpful to have a framework that the community can use, fix and improve so that each integrator doesn't have to reinvent these solutions.

The Basic Design

The basic design of the tika-batch module is intended for conventional processing – if anything can be reused/modified for hadoop, please contribute!

The overall design is a producer/consumer design pattern. The producer and consumers share an ArrayBlockingQueue.

Given a wide range of use cases for Tika, it would be great if the batch process were highly configurable. The current implementation includes an xml config file and relies on builders. This will allow developers to add their own components into the current framework as long as they also include builders.

Tika-batch has far more code than I originally envisioned, but multi-threading, multi-processing, robust logging and configurability are common culprits for code-bloat. Any input into code-pruning is welcome!

BatchProcess

A BatchProcess manages a single process for: a ResourceCrawler, ResourceConsumers, an Interrupter and a StatusReporter. Each ResourceConsumer runs in its own thread, and the user can specify the number of consumer threads.

ResourceCrawler

Generically, a ResourceCrawler adds potential files for processing onto the queue. The initial implementation of tika-batch offers two: one crawls a file system directory, and one reads a list of files to be processed in a file system directory.

Some other implementations of a ResourceCrawler might include a directory listener or a database exporter. Anything else?

ResourceConsumers

A ResourceConsumer pulls a resource from the queue and consumes it. A resource should be a lightweight pointer to a file resource (not the actual bytes!), and it returns an InputStream and a Metadata object.

Interrupter

An interrupter runs in a separate thread and allows users to ask the BatchProcess to shut down gracefully.

StatusReporter

A StatusReporter runs in a separate thread. It has visibility into the crawler and the consumers, and it periodically reports on how many files have been processed, how many exceptions, and so on.

StaleChecker

This is an inner class within BatchProcess that periodically checks for stale consumers. It will cause the BatchProcess to shut down if it finds a stale consumer. In earlier versions of the code, the user could specify the maximum number of stale threads before BatchProcess shutdown, however, in practice, a single stale consumer can tie up a huge amount of resources. For now, BatchProcess will shutdown if it finds one stale consumer.

ProcessDriver

This initiates the BatchProcess and monitors it to make sure that it is still alive when it should be. If the BatchProcess sends a restart signal via stderr or a restart exit code to the ProcessDriver, the ProcessDriver will restart the BatchProcess.

File System (FS) Batch, Step 1

The initial use case for tika-batch is to process a directory of files recursively and generate an output file for each input file. The output directory has the same structure/hierarchy as the input folder, and each output file has a file suffix appended to it depending on the ContentHandler (".xml", ".txt", ".json", etc).

FS Resource Crawlers

For FSBatch, the directory crawler starts with a root directory and crawls all files. Bells and whistles include:

- The directory crawler can start with a root directory and a file list, and it will "crawl" all of the files on the list relative to the root directory. This is very useful for testing or for processing a subset of documents.
- The directory crawler uses a Tika DocumentSelector to determine whether or not to add a file to the queue. The only metadata available to the directory crawler at this point in the processing is the file name and the length of the file in bytes. The user can specify a regex for files to include (based on filename) and a regex for files to exclude (based on filename); the user can also specify a max bytes limit.
- The directory crawler also has the idea of a start directory. This is a child directory of the root directory. This allows users another way to process a subset of a directory structure.

I've also added a strawman driver that runs multiple threads but kicks off a single app.jar process for every file.

FS Resource Consumers

The tika-batch package includes an abstract ResourceConsumer class that handles much of the multi-threading burden. Concrete classes of resource consumer only have to implement processFileResource(FileResource fileResource). Concrete classes should also handle all exceptions that they want to handle and make appropriate calls to incrementHandledExceptions().

There are two consumers currently.

- One handles traditional Tika processing with the ToTextContentHandler, the ToHtmlContentHandler or the ToXMLContentHandler. The user can specify a write limit and whether or not to process documents recursively.
- The other offers handling by the (to be added) RecursiveParserWrapper. For each input file, the output is a json-formatted list of Metadata items, with a special key for the content.

FSResourceConsumers rely on a ContentHandlerFactory to get the user-specified handler and an OutputStreamFactory to get the FileOutputStream to write to.

ContentHandlerFactory

This is a simple class that builds a handler for the three basic types mentioned above the ToXXXContentHandler and optionally specifies a write limit.

OutputStreamFactory

This calculates the output (target) file location and name, builds the requisite parent directories and returns the OutputStream to the consumer.

If a target file exists, this will do one of three things:

- Skip it – return a null OutputStream. The consumers know to avoid parsing a file if the returned OutputStream is null.
- Rename the file – add e.g. (1) to the end of the file name (before the suffix) until there is a new file.
- Overwrite the existing file.

USAGE

See TikaBatchUsage.

TODO

Any design recommendations at this point? See TIKA-1330.

Tika Server Client

Tika Batch Hadoop

See TikaInHadoop.