Io

IO Package

The "io" package contains a number of i/o-centric utility classes used by other packages within Hadoop. There are three files, however, that are especially interesting to consider and reuse:

- UTF8.java
- SequenceFile.java
- MapFile.java

UTF8

OK, UTF8 isn't actually the most interesting class in the world. It's a string, with fewer methods than the Java class lang.String. However, UTF8 has a few advantages. The biggest is that it uses the UTF8 compressed encoding for Unicode. When you have an application that stores an enormous number of strings, the memory savings can add up.

Also note that UTF8 is mutable, unlike the Java string class, so it can be reused if needed.

SequenceFile

SequenceFile is a simple linear list of key/value pairs. Writers can only add to the end of a SequenceFile. The keys and values must both implement org.apache.hadoop.io.Writable.

SequenceFile.Reader reads in an existing SequenceFile. Readers can call next() repeatedly to iterate through the file, or can seek to a given byte position.

SequenceFile.Writer writes to a new SequenceFile. Writers can only append to the end of a file.

SequenceFile.Sorter reads in an existing SequenceFile and sorts the entries according to the key values. The key class of the SequenceFile must implement WritableComparable in order for the Sorter to work. Sorter can sort a file of any size, using temporary on-disk files if necessary. This page has a good description of how external sorting works.

Because many text and search tasks are both batch-oriented and much larger than available memory, it's often useful to decompose tasks as a series of external sort operations.

MapFile

MapFile adds functionality to a SequenceFile. It also stores key-value pairs on disk, but unlike SequenceFile allows for efficient random access. It is implemented by storing both a SequenceFile and an associated index file. The small index file is kept in memory, while the much larger SequenceFile is looked up as necessary.

MapFile.Reader can either step linearly through the file or can seek to an arbitrary key value location.

MapFile.Writer adds items to the file. Keys must implement WritableComparable, and keys must be added in monotonically increasing order. Often, that means the set of key/values must be first sorted with SequenceFile.Sorter, and then appended to the MapFile.Writer.

Note that while a single MapFile lookup is fast, a series of arbitrary lookups probably won't be (as each lookup can involve a disk seek to fetch the target item). If you have a large number of key/value pairs to process via MapFile lookups, it's probably better to use the "sort-merge-join" technique, which you can perform using a series of sorted SequenceFiles.