NutchFileFormats

Introduction

The page provides information on the Nutch file formats (for the Nutch 1.X series) from the bottom up. Within the context of this document, we use the terminology **custom types** to refer to physical files which can be written by Nutch. More is explained about Writable's below. **N.B.** Nutch implements several core data structures and serialization mechanisms directly from Apache Hadoop so please read this document with that in mind.

Nutch CustomWritable's

Nutch implements its own custom serialization to store custom serialized Java data types and structures on file. The interface org.apache.hadoop.io.Writable must be implemented for all such data types.

The list below indicates all of the Nutch custom writable's which implement the Hadoop org.apache.hadoop.io.Writable interface. The remaining sections of this page explains how and where each of these Writable's fits into the core Nutch data structures such are **CrawlDB**, **LinkDB** and **Segments**.

```
./src/java/org/apache/nutch/crawl/CrawlDatum.java
./src/java/org/apache/nutch/crawl/Generator.java
./src/java/org/apache/nutch/crawl/Inlink.java
./src/java/org/apache/nutch/crawl/Inlinks.java
./src/java/org/apache/nutch/crawl/MapWritable.java
./src/java/org/apache/nutch/indexer/NutchDocument.java
./src/java/org/apache/nutch/indexer/NutchField.java
./src/java/org/apache/nutch/indexer/NutchIndexAction.java
./src/java/org/apache/nutch/metadata/Metadata.java
./src/java/org/apache/nutch/parse/Outlink.java
./src/java/org/apache/nutch/parse/ParseStatus.java
./src/java/org/apache/nutch/parse/ParseText.java
./src/java/org/apache/nutch/protocol/Content.java
./src/java/org/apache/nutch/protocol/ProtocolStatus.java
./src/java/org/apache/nutch/scoring/webgraph/LinkDatum.java
./src/java/org/apache/nutch/scoring/webgraph/LinkDumper.java
./src/java/org/apache/nutch/scoring/webgraph/Loops.java
./src/java/org/apache/nutch/scoring/webgraph/Node.java
```

With the above in mind, lets look at the composite features of some of these custom Writable's

Writable Composition

org.apache.hadoop.io.Text

Nutch uses Java's native UTF-8 character set, and the class org.apache.hadoop.io.Text for writing short strings to files. The UTF8 class limits the length of strings to 0xffff/3 or 21845 bytes. The function UTF8.write() uses java.io.DataOutput.writeShort() to prepend the length of the string. This is why the two bytes \000\003 is seen before a three letter word in a file. The zero byte is thus not a null termination of the previous string (strings are not null terminated), but the most significant byte of the 16 bit short integer indicating the length of the following string.

org.apache.hadoop.io.SequenceFile

Nutch relies heavily on mappings (associative arrays) from keys to values. The class SequenceFile is a flat file of keys and values. The first four bytes of each such file are ASCII "SEQ" and \001 (C-a), followed by the Java class names of keys and values, written as UTF8 strings, e.g. "SEQ\001\000\004long\000\004long", for a mapping from long integers to long integers. After that follows the key-value pairs. Each pair is introduced by four bytes telling the length in bytes of the pair (excluding the eight length bytes) and four bytes telling the length of the key. The typical long (64 bit) integer is 8 bytes and a long-to-long mapping will have pairs of length 16 bytes, e.g.

```
  00 00 00 10                                 int length of pair = 0x10 = 16 bytes
  00 00 00 08                                 int length of key  = 0x08 =  8 bytes
  00 00 00 00 00 00 02 80       long key = 0x280 = 640
  00 00 00 00 00 0a 42 9b       long value = 0xa429b = 672411
```

Much more on SequenceFile characteristics such as file headers and compression options can be found at the SequenceFile Javadoc.

org.apache.hadoop.io.MapFile

To economize the handling of large data volumes, MapFile manages a mapping as two separate files in a subdirectory of its own. The large "data" file stores all keys and values, sorted by the key. The much smaller "index" file points to byte offsets in the data file for a small sample of keys. Only the index file is read into memory.

ArrayFile is a specialization of MapFile, specifically a dense file-based mapping from integers to values where the keys are long integers. Finally you can also see SetFile which is a file representing a file-based set of keys.

Additional files in org.apache.hadoop.io.* package contains the actual Writer, Reader and Sorter implementations as well.

CrawlDB

Description

Nutch maintains a CrawlDB containing CrawlDatum objects.

Directory Structure

```
.
 current
    part-00000
        data
        index
 old
     part-00000
        data
        index
     part-00001
         data
         index
     ...
```

File Formats

It is advised that you follow the Javadoc links within the table to get a better understanding of the data types.

file	key datatype	value datatype	codec
data	org.apache.hadoop.io.Text	org.apache.nutch.crawl.CrawlDatum	org.apache.hadoop.io.compress.DefaultCodec
index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec

LinkDB

Description

Maintains an inverted link map, listing incoming links for each url.

Directory Structure

```
.
 current
     part-00000
         data
         index
```

File Formats

N.B. It is advised that you follow the Javadoc links within the table to get a better understanding of the data types.

file	key datatype	value datatype	codec
data	org.apache.hadoop.io.Text	org.apache.nutch.crawl.Inlinks	org.apache.hadoop.io.compress.DefaultCodec
index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec

Segments

Description

When Nutch crawls the web, each resulting segment (segments contain the actual content which was fetched) has four subdirectories, each containing an ArrayFile (a MapFile having keys that are long integers).

Directory Structure

```
.
 content
    part-00000
       data
       index
    part-...
 crawl_fetch
    part-00000
       data
       index
    part-...
 crawl_generate
    part-00000
 crawl_parse
    part-00000
    part-00001
 parse_data
    part-00000
       data
       index
    part-...
 parse_text
     part-00000
        data
        index
     part-...
```

File Formats

N.B. It is advised that you follow the Javadoc links within the table to get a better understanding of the data types.

directory	file	key datatype	value datatype	codec
content	data	org.apache.hadoop.io.Text	org.apache.nutch.protocol.Content	org.apache.hadoop.io.compress.DefaultCodec
content	index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec
crawl_fetch	data	org.apache.hadoop.io.Text	org.apache.nutch.crawl.CrawlDatum	org.apache.hadoop.io.compress.DefaultCodec
crawl_fetch	index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec
crawl_generate	part-0000	org.apache.hadoop.io.Text	org.apache.nutch.crawl.CrawlDatum	org.apache.hadoop.io.compress.DefaultCodec
crawl_parse	data	org.apache.hadoop.io.Text	org.apache.nutch.crawl.CrawlDatum	org.apache.hadoop.io.compress.DefaultCodec
crawl_parse	index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec
parse_data	data	org.apache.hadoop.io.Text	org.apache.nutch.parse.ParseData	org.apache.hadoop.io.compress.DefaultCodec
parse_data	index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec
parse_text	data	org.apache.hadoop.io.Text	org.apache.nutch.parse.ParseText	org.apache.hadoop.io.compress.DefaultCodec
parse_text	index	org.apache.hadoop.io.Text	org.apache.hadoop.io.LongWritable	org.apache.hadoop.io.compress.DefaultCodec

Old File Format Documentation

Nutch version 0.5

-- Differences noted by MattKangas - 04 Jan 2005

A segment now consists of five subdirectories, each containing an ArrayFile:

{{{#!CSV , Subdirectory,Value datatype,Variable fetchlist,net.nutch.pagedb.FetchListEntry,fetchList
fetcher,net.nutch.fetcher.FetcherOutput,fetcherWriter content,net.nutch.protocol.Content,contentWriter
parse_text,net.nutch.parse.ParseText,parseTextWriter parse_data,net.nutch.parse.ParseData,parseDataWriter
}}}

FetcherOutput is changed:

```
1 byte version (value 4, was 3)
FetchListEntry as specified above
16 bytes MD5 hash
1 byte status
8 bytes (long) Java milliseconds fetchdate
```

New class: net.nutch.protocol.Content

```
1 byte version (value 1)
UTF8 string url
UTF8 string base
compressed byte array content
UTF8 string contentType
java.util.Properties metadata
```

New class: net.nutch.parse.ParseText

```
1 byte version (value 1)
compressed byte array text
```

New class: net.nutch.parse.ParseData

```
1 byte version (value 1)
UTF8 string title
4 bytes integer totalOutlinks
a list of net.nutch.fetcher.Outlink objects:
        UTF8 string URL
        UTF8 string anchor
java.util.Properties metadata
```

Nutch version 0.4

Nutch 0.4 was released on May 25, 2004 (the previous version 0.3 was from June 17, 2003). The Java source code consists of 165 files comprising 37,178 lines of code.

Nutch implements its own serialization to store serialized Java data types and structures on file. The interface net.nutch.io.Writable must be implemented for all such data types. In some cases, long text strings are stored in GZIP (Gnu ZIP) compressed format.

The abstract class nutch.io.VersionedWritable prepends a byte indicating the version of the data structure, typically \001.

Nutch uses Java's native UTF-8 character set, and the class net.nutch.io.UTF8 for writing short strings to files. The UTF8 class limits the length of strings to 0xffff/3 or 21845 bytes. The function UTF8.write() uses java.io.DataOutput.writeShort() to prepend the length of the string. This is why the two bytes \000\003 is seen before a three letter word in a file. The zero byte is thus not a null termination of the previous string (strings are not null terminated), but the most significant byte of the 16 bit short integer indicating the length of the following string.

Nutch relies heavily on mappings (associative arrays) from keys to values. The class net.nutch.io.SequenceFile is a flat file of keys and values. The first four bytes of each such file are ASCII "SEQ" and \001 (C-a), followed by the Java class names of keys and values, written as UTF8 strings, e.g. "SEQ\001\000\004long\000\004long", for a mapping from long integers to long integers. After that follows the key-value pairs. Each pair is introduced by four bytes telling the length in bytes of the pair (excluding the eight length bytes) and four bytes telling the length of the key. The typical long (64 bit) integer is 8 bytes and a long-to-long mapping will have pairs of length 16 bytes, e.g.

```
00 00 00 10                                 int length of pair = 0x10 = 16 bytes
00 00 00 08                                 int length of key  = 0x08 =  8 bytes
00 00 00 00 00 00 02 80       long key = 0x280 = 640
00 00 00 00 00 0a 42 9b       long value = 0xa429b = 672411
```

To economize the handling of large data volumes, net.nutch.io.MapFile manages a mapping as two separate files in a subdirectory of its own. The large "data" file stores all keys and values, sorted by the key. The much smaller "index" file points to byte offsets in the data file for a small sample of keys. Only the index file is read into memory.

net.nutch.io.ArrayFile is a specialization of MapFile where the keys are long integers.

The Java files in net.nutch.io.* comprise 2556 lines of source code. The biggest one is Sequencefile.java, which contains a Writer (112 lines), a Reader (138 lines), a BufferedRandomAccessFile (140 lines) and a Sorter (389 lines).

When Nutch crawls the web, each resulting segment has four subdirectories, each containing an ArrayFile (a MapFile having keys that are long integers):

{{{#!CSV , Subdirectory,Value datatype,Variable fetchlist,net.nutch.pagedb.FetchListEntry,fetchList
fetcher,net.nutch.fetcher.FetcherOutput,fetcherDb fetcher_content,net.nutch.fetcher.FetcherContent,rawDb
fetcher_text,net.nutch.fetcher.FetcherText,strippedDb
}}}

Crawling is performed by net.nutch.fetcher.Fetcher which starts a number of parallel FetcherThread?. Each thread gets an URL from the fetchList, checks robots.txt, retrieves the contents and appends the results to fetcherDb, rawDb, and strippedDb.

The FetchListEntry is represented thus:

```
1 byte version (value should be 2),
1 byte flag (value 1 = true if page should be fetched)
page, as defined by net.nutch.db.Page:
       1 byte version (value should be 4)
       UTF8 string URL (primary key)
       16 bytes (128 bit) MD5 hash of contents
       8 bytes (64 bit long) Java milliseconds when page should be refetched
       1 byte number of failed attempts
       1 byte fetch interval in days
       4 bytes Java float score
       4 bytes Java float next score
4 bytes number of anchors
a list of anchors represented as UTF8 strings
```

The FetcherOutput is all of the fetcher's output except the raw and stripped versions of the contents:

```
1 byte version (value 3)
FetchListEntry as specified above
16 bytes MD5 hash
1 byte status
UTF8 string title
4 bytes integer totalOutlinks
a list of net.nutch.fetcher.Outlink objects:
        UTF8 string URL
        UTF8 string anchor
8 bytes (long) Java milliseconds fetchdate
```

The FetcherContent is the raw contents stored in GZIP:

```
1 byte version (value 1)
compressed byte array
```

The FetcherText is the text conversion of page's content, stored in GZIP:

```
1 byte version (value 1)
compressed byte array
```