Package org.htmlparser.lexer

The lexer package is the base level I/O subsystem.

See:
Description

Class Summary

Cursor	A bookmark in a page.
InputStreamSource	A source of characters based on an InputStream such as from a URLConnection.
Lexer	This class parses the HTML stream into nodes.
Page	Represents the contents of an HTML page.
PageAttribute	An attribute within a tag on a page.
PageIndex	A sorted array of integers, the positions of the first characters of each line.
Source	A buffered source of characters.
Stream	Provides for asynchronous fetching from a stream.
StringSource	A source of characters based on a String.

Package org.htmlparser.lexer Description

The lexer package is the base level I/O subsystem.

The lexer package is responsible for reading characters from the HTML source and identifying the node lexemes. For example, the HTML code below would return the list of nodes shown:

```
<html><head><title>Humoresque</title></head>
<body bgcolor='silver'>
Passengers will please refrain
from flushing toilets while the train
is standing in the station. I love you!
<p>
We encourage constipation
while the train is in the station
If the train can't go
then why should you.
</body>
</html>
```

1. line 0, offset 0, to line 0, offset 6, html tag
2. line 0, offset 6, to line 0, offset 12, head tag
3. line 0, offset 12, to line 0, offset 19, title tag

4. line 0, offset 19, to line 0, offset 29, string node "Humoresque"
5. line 0, offset 29, to line 0, offset 37, end title tag
6. line 0, offset 37, to line 0, offset 44, end head tag
7. line 0, offset 44, to line 0, offset 45, string node "\n"
8. line 1, offset 0, to line 1, offset 23, body tag
9. line 1, offset 23, to line 4, offset 40, string node "\nPassengers...you!\n"
10. line 5, offset 0, to line 5, offset 2, paragraph tag
11. line 5, offset 3, to line 9, offset 21, string node "\nWe...you.\n"
12. line 10, offset 0, to line 10, offset 7, end body tag
13. line 10, offset 8, to line 10, offset 9, string "\n"
14. line 11, offset 0, to line 11, offset 7, html tag
15. line 11, offset 7, to line 11, offset 8, string node "\n"

Stream, Source, Page and Lexer

The package is arranged in four levels, `Stream`, `Source` `Page` and `Lexer` in the order of lowest to highest. A `Stream` is raw bytes from the URLConnection or file. It has no intelligence. A `Source` is raw characters, hence it knows about the encoding scheme used and can be reset if a different encoding is detected after partially reading in the text. A `Page` provides characters from the source while maintaining the index of line numbers, and hence can be thought of as an array of strings corresponding to source file lines, but it doesn't actually store any text, relying on the buffering within the `Source` instead. The `Lexer` contains the actual lexeme parsing code. It reads characters from the page, keeping track of where it is with a `Cursor` and creates the array of nodes using various state machines.

The following are some design goals and 'invariants' within the package, if you are attempting to understand or modify it.

Contiguous Nodes
: Adjacent nodes have no characters between them. The list of nodes forms an uninterrupted chain that, by start and end definitions, completely covers the characters that were read from the HTML source.

Text Fidelity
: Besides complete coverage, the nodes do not initially contain copies of the text, but instead simply contain offsets into a single large buffer that contains the text read from the HTML source. Even within tags, the attributes list can contain whitespace, thus there is no lost whitespace or text formatting either outside or within tags. Upper and lower case text is preserved.

Line Endings
: End of line characters are just whitespace. There is no distinction made between end of line characters (or pairs of characters on Windows) and other whitespace. The text is not read in line by line so nodes (tags) can easily span multiple lines with no special processing. Line endings are not transformed between platforms, i.e. Unix line endings are not converted to Windows line endings by this level. Each node has a starting and ending location, which the page can use to extract the text. To facilitate formatting error and log messages the page can turn these offsets into row and column numbers. In general ignore line breaks in the source if at all possible.

State Machines
: The Lexer has the following state machines:

- in text - parseString()
- in comment - parseRemark()
- in tag - parseTag()
- in JSP tag - parseJsp()

There is another state machine -- parseCDATA -- used by higher level code (script and style scanners), but this isn't actually used by the lexer.

Two Jars

For elementary operations at the node level, a minimalist jar file containing only the lexer and base tag classes is split out from the larger `htmlparser.jar`. In this way, simple parsing and output is handled with a jar file that is under 45 kilobytes, but anything beyond peephole manipulation, i.e. closing tag detection and other semantic reasoning, will need the full set of scanners, nodes and ancillary classes, which now stands at 210 kilobytes.

© 2006 Derrick Oswald
Sep 17, 2006

HTML Parser is an open source library released under Common Public License.