Creak: A Swift HTML Parsing Library

Creak is designed to parse HTML documents efficiently and build a tree structure representing the document's elements. The parsing process involves several key steps and components that work together to achieve this goal. Here's a detailed explanation of how Creak parses HTML:

Parsing Process Overview

1. **Initialization**: The HTML string is loaded and cleaned.
2. **Tokenization**: The HTML string is broken down into tokens representing different parts of the HTML, such as tags and text.
3. **Tree Construction**: The tokens are used to construct a tree structure of nodes, representing the HTML document's elements and text.

Key Components

- **Dom Class**: Manages the overall parsing process and stores the root of the parsed HTML tree.
- **Content Class**: Provides utility functions for tokenizing the HTML string.
- **HtmlNode and TextNode Classes**: Represent the elements and text nodes in the HTML document.
- **Tag Class**: Represents HTML tags and their attributes.

Detailed Parsing Steps

1. Initialization The `Dom` class is responsible for initializing the parsing process. The `loadStr` method takes the raw HTML string, cleans it, and initializes the `Content` object.

```
public func loadStr(str: String) -> Dom {
    raw = str
    let html = clean(str)
    content = Content(content: html)
    parse()
    return self
}
```

2. Tokenization The `Content` class provides utility functions to tokenize the HTML string. It includes methods to copy sections of the string, skip characters, and handle tokens such as tags and attributes.

- **copyUntil**: Copies characters from the current position until a specified character is encountered.
- **skipByToken**: Skips characters based on a specified token.

These methods are used to identify and extract different parts of the HTML, such as tags, attributes, and text content.

3. Tree Construction The `parse` method in the `Dom` class iterates through the HTML string, identifying tags and text, and building a tree structure of nodes (`HtmlNode` and `TextNode`).

```
private func parse() {
    root = HtmlNode(tag: "root")
    var activeNode: InnerNode? = root
    while activeNode != nil {
        let str = content.copyUntil("<")
        if (str == "") {
            let info = parseTag()
            if !info.status {
                activeNode = nil
                continue
            }

            if info.closing {
                let originalNode = activeNode
                while activeNode?.tag.name != info.tag {
                    activeNode = activeNode?.parent
                    if activeNode == nil {
                        activeNode = originalNode
                        break
                    }
                }
                if activeNode != nil {
                    activeNode = activeNode?.parent
                }
                continue
            }

            if info.node == nil {
                continue
            }

            let node = info.node!
            activeNode!.addChild(node)
            if !node.tag.selfClosing {
```

```
                activeNode = node
            }
        } else if (trim(str) != "") {
            let textNode = TextNode(text: str)
            activeNode?.addChild(textNode)
        }
    }
}
```

- **Root Node**: The parsing starts with a root node (`HtmlNode` with tag "root").
- **Active Node**: The `activeNode` variable keeps track of the current node being processed.
- **Text Content**: If text content is found, a `TextNode` is created and added to the current node.
- **Tag Parsing**: If a tag is found, the `parseTag` method is called to handle it.

Parsing Tags The `parseTag` method handles the identification and processing of tags.

```
private func parseTag() -> ParseInfo {
    var result = ParseInfo()
    if content.char() != ("<" as Character) {
        return result
    }

    if content.fastForward(1).char() == "/" {
        var tag = content.fastForward(1).copyByToken(Content.Token.Slash, char: true)
        content.copyUntil(">")
        content.fastForward(1)

        tag = tag.lowercaseString
        if selfClosing.contains(tag) {
            result.status = true
            return result
        } else {
            result.status = true
            result.closing = true
            result.tag = tag
            return result
        }
    }
```

```
let tag = content.copyByToken(Content.Token.Slash, char: true).lowercaseString
let node = HtmlNode(tag: tag)

while content.char() != ">" &&
   content.char() != "/" {
    let space = content.skipByToken(Content.Token.Blank, copy: true)
    if space?.characters.count == 0 {
        content.fastForward(1)
        continue
    }

    let name = content.copyByToken(Content.Token.Equal, char: true)
    if name == "/" {
        break
    }

    if name == "" {
        content.fastForward(1)
        continue
    }

    content.skipByToken(Content.Token.Blank)
    if content.char() == "=" {
        content.fastForward(1).skipByToken(Content.Token.Blank)
        var attr = AttrValue()
        let quote: Character? = content.char()
        if quote != nil {
            if quote == "\"" {
                attr.doubleQuote = true
            } else {
                attr.doubleQuote = false
            }
            content.fastForward(1)
            var string = content.copyUntil(String(quote!), char: true, escape: true)
            var moreString = ""
```

```
                repeat {
                    moreString = content.copyUntilUnless(String(quote!), unless: "=>")
                    string += moreString
                } while moreString != ""
                attr.value = string
                content.fastForward(1)
                node.setAttribute(name, attrValue: attr)
            } else {
                attr.doubleQuote = true
                attr.value = content.copyByToken(Content.Token.Attr, char: true)
                node.setAttribute(name, attrValue: attr)
            }
        } else {
            node.tag.setAttribute(name, attrValue: AttrValue(nil, doubleQuote: true))
            if content.char() != ">" {
                content.rewind(1)
            }
        }
    }

    content.skipByToken(Content.Token.Blank)
    if content.char() == "/" {
        node.tag.selfClosing = true
        content.fastForward(1)
    } else if selfClosing.contains(tag) {
        node.tag.selfClosing = true
    }

    content.fastForward(1)

    result.status = true
    result.node = node

    return result
}
```

- **Tag Identification**: The method identifies whether a tag is an opening or closing tag.

- **Attributes**: It parses the attributes of the tag and adds them to the `HtmlNode`.
- **Self-closing Tags**: It handles self-closing tags appropriately.

Conclusion

Creak' s parsing process involves initializing the HTML content, tokenizing it, and constructing a tree structure of nodes. The `Dom` class manages the overall parsing, while the `Content` class provides utilities for tokenizing the HTML string. The `HtmlNode` and `TextNode` classes represent the elements and text in the HTML document, and the `Tag` class manages the attributes of the tags. This efficient and organized approach makes Creak a powerful tool for parsing HTML in Swift.