Creak：一個 Swift 的 HTML 解析庫

Creak 設計用於高效解析 HTML 文檔，並構建表示文檔元素的樹結構。解析過程涉及多個關鍵步驟和組件，這些組件協同工作以實現此目標。以下是 Creak 解析 HTML 的詳細說明：

解析過程概述

1. **初始化**：加載並清理 HTML 字符串。
2. **標記化**：將 HTML 字符串分解為表示 HTML 不同部分的標記，例如標籤和文本。
3. **樹結構構建**：使用標記構建表示 HTML 文檔元素和文本的樹結構。

關鍵組件

- **Dom 類**：管理整個解析過程並存儲解析後的 HTML 樹的根節點。
- **Content 類**：提供標記化 HTML 字符串的實用函數。
- **HtmlNode 和 TextNode 類**：表示 HTML 文檔中的元素和文本節點。
- **Tag 類**：表示 HTML 標籤及其屬性。

詳細解析步驟

1. 初始化 `Dom` 類負責初始化解析過程。`loadStr` 方法接受原始 HTML 字符串，對其進行清理，並初始化 `Content` 對象。

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

2. 標記化 `Content` 類提供用於標記化 HTML 字符串的實用函數。它包括從當前字符位置複製字符、跳過字符以及處理標籤和屬性等標記的方法。

- **copyUntil**：從當前位置複製字符，直到遇到指定字符。
- **skipByToken**：根據指定的標記跳過字符。

這些方法用於識別和提取 HTML 的不同部分，例如標籤、屬性和文本內容。

3. 樹結構構建 `Dom` 類中的 `parse` 方法遍歷 HTML 字符串，識別標籤和文本，並構建由 `HtmlNode` 和 `TextNode` 組成的樹結構。

```
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)
        }
    }
}
```

- **根節點**：解析從根節點 (`HtmlNode`，標籤為 “root”) 開始。
- **活動節點**：`activeNode` 變量跟踪當前處理的節點。
- **文本內容**：如果發現文本內容，會創建一個 `TextNode` 並添加到當前節點。
- **標籤解析**：如果發現標籤，會調用 `parseTag` 方法處理它。

標籤解析 `parseTag` 方法處理標籤的識別和處理。

```
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
}
```

- **標籤識別**：該方法識別標籤是開標籤還是閉標籤。
- **屬性**：解析標籤的屬性並將其添加到 `HtmlNode`。
- **自閉合標籤**：適當地處理自閉合標籤。

結論

Creak 的解析過程涉及初始化 HTML 內容、將其標記化並構建節點的樹結構。`Dom` 類管理整體解析，而 `Content` 類提供標記化 HTML 字符串的實用函數。`HtmlNode` 和 `TextNode` 類表示 HTML 文檔中的元素和文本，`Tag` 類管理標籤的屬性。這種高效且有組織的方法使 Creak 成為 Swift 中解析 HTML 的強大工具。