Creak: 一个 Swift 的 HTML 解析库

Creak 设计用于高效解析 HTML 文档，并构建表示文档元素的树结构。解析过程涉及多个关键步骤和组件，这些组件协同工作以实现此目标。以下是 Creak 解析 HTML 的详细说明：

解析过程概述

1. **初始化**：加载并清理 HTML 字符串。
2. **标记化**：将 HTML 字符串分解为表示 HTML 不同部分的标记，例如标签和文本。
3. **树结构构建**：使用标记构建表示 HTML 文档元素和文本的树结构。

关键组件

- **Dom 类**：管理整个解析过程并存储解析后的 HTML 树的根节点。
- **Content 类**：提供标记化 HTML 字符串的实用函数。
- **HtmlNode 和 TextNode 类**：表示 HTML 文档中的元素和文本节点。
- **Tag 类**：表示 HTML 标签及其属性。

详细解析步骤

1. 初始化 `Dom` 类负责初始化解析过程。`loadStr` 方法接受原始 HTML 字符串，对其进行清理，并初始化 `Content` 对象。

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

2. 标记化 `Content` 类提供用于标记化 HTML 字符串的实用函数。它包括从当前字符位置复制字符、跳过字符以及处理标签和属性等标记的方法。

- **copyUntil**：从当前位置复制字符，直到遇到指定字符。
- **skipByToken**：根据指定的标记跳过字符。

这些方法用于识别和提取 HTML 的不同部分，例如标签、属性和文本内容。

3. 树结构构建 `Dom` 类中的 `parse` 方法遍历 HTML 字符串，识别标签和文本，并构建由 `HtmlNode` 和 `TextNode` 组成的树结构。

```
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)
        }
    }
}
```

- **根节点**：解析从根节点 (`HtmlNode`，标签为“root”) 开始。
- **活动节点**：`activeNode` 变量跟踪当前处理的节点。
- **文本内容**：如果发现文本内容，会创建一个 `TextNode` 并添加到当前节点。
- **标签解析**：如果发现标签，会调用 `parseTag` 方法处理它。

标签解析 `parseTag` 方法处理标签的识别和处理。

```
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
}
```

- **标签识别**：该方法识别标签是开标签还是闭标签。
- **属性**：解析标签的属性并将其添加到 `HtmlNode`。
- **自闭合标签**：适当地处理自闭合标签。

结论

Creak 的解析过程涉及初始化 HTML 内容、将其标记化并构建节点的树结构。`Dom` 类管理整体解析，而 `Content` 类提供标记化 HTML 字符串的实用函数。`HtmlNode` 和 `TextNode` 类表示 HTML 文档中的元素和文本，`Tag` 类管理标签的属性。这种高效且有组织的方法使 Creak 成为 Swift 中解析 HTML 的强大工具。