Creak : Une bibliothèque d’’analyse HTML pour Swift

Creak est conçu pour analyser efficacement les documents HTML et construire une structure arborescente représentant les éléments du document. Le processus d’analyse implique plusieurs étapes et composants clés qui travaillent ensemble pour atteindre cet objectif. Voici une explication détaillée de la manière dont Creak analyse le HTML :

Aperçu du processus d’analyse

1. **Initialisation** : Chargement et nettoyage de la chaîne HTML.
2. **Tokenisation** : Décomposition de la chaîne HTML en tokens représentant les différentes parties du HTML, telles que les balises et le texte.
3. **Construction de l’arbre** : Utilisation des tokens pour construire une structure arborescente représentant les éléments et le texte du document HTML.

Composants clés

- **Classe Dom** : Gère l’ensemble du processus d’analyse et stocke le nœud racine de l’arbre HTML analysé.
- **Classe Content** : Fournit des fonctions utilitaires pour la tokenisation des chaînes HTML.
- **Classes HtmlNode et TextNode** : Représentent les éléments et les nœuds de texte dans un document HTML.
- **Classe Tag** : Représente les balises HTML et leurs attributs.

Étapes détaillées d’analyse

1. Initialisation La classe `Dom` est responsable de l’initialisation du processus d’analyse. La méthode `loadStr` prend une chaîne HTML brute, la nettoie et initialise l’objet `Content`.

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

2. Tokenisation La classe `Content` fournit des fonctions utilitaires pour la tokenisation de chaînes HTML. Elle inclut des méthodes pour copier des caractères à partir de la position actuelle, sauter des caractères, ainsi que pour traiter des tokens tels que les balises et les attributs.

- **copyUntil** : Copie les caractères à partir de la position actuelle jusqu'à ce qu'un caractère spécifié soit rencontré.
- **skipByToken** : Saute les caractères en fonction du jeton spécifié.

Ces méthodes sont utilisées pour identifier et extraire différentes parties du HTML, telles que les balises, les attributs et le contenu textuel.

3. Construction de la structure en arbre La méthode `parse` de la classe `Dom` parcourt la chaîne HTML, identifie les balises et le texte, et construit une structure arborescente composée de `HtmlNode` et `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)
        }
    }
}
```

- **Nœud racine** : L'analyse commence à partir du nœud racine (`HtmlNode`, avec la balise "root").
- **Nœud actif** : La variable `activeNode` suit le nœud en cours de traitement.
- **Contenu textuel** : Si du contenu textuel est détecté, un `TextNode` est créé et ajouté au nœud actuel.
- **Analyse des balises** : Si une balise est détectée, la méthode `parseTag` est appelée pour la traiter.

Analyse des étiquettes La méthode `parseTag` gère l'identification et le traitement des balises.

```
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
}
```

- **Reconnaissance des balises** : Cette méthode identifie si une balise est une balise ouvrante ou fermante.
- **Attributs** : Les attributs de la balise sont analysés et ajoutés au `HtmlNode`.
- **Balises auto-fermantes** : Les balises auto-fermantes sont gérées de manière appropriée.

Conclusion

Le processus d'analyse de Creak implique l'initialisation du contenu HTML, sa tokenisation et la construction d'une structure arborescente de nœuds. La classe `Dom` gère l'analyse globale, tandis que la classe `Content` fournit des fonctions utilitaires pour la tokenisation des chaînes HTML. Les classes `HtmlNode` et `TextNode` représentent respectivement les éléments et le texte dans un document HTML, et la classe `Tag` gère les attributs des balises. Cette approche efficace et organisée fait de Creak un outil puissant pour l'analyse HTML en Swift.