Creak: Una biblioteca de análisis de HTML en Swift

Creak está diseñado para analizar eficientemente documentos HTML y construir una estructura de árbol que representa los elementos del documento. El proceso de análisis involucra varios pasos y componentes clave que trabajan en conjunto para lograr este objetivo. A continuación, se presenta una explicación detallada de cómo Creak analiza el HTML:

Resumen del Proceso de Análisis

1. **Inicialización**: Cargar y limpiar la cadena HTML.
2. **Tokenización**: Descomponer la cadena HTML en tokens que representan diferentes partes del HTML, como etiquetas y texto.
3. **Construcción del árbol**: Utilizar los tokens para construir una estructura de árbol que represente los elementos y el texto del documento HTML.

Componentes Clave

- **Clase Dom**: Gestiona todo el proceso de análisis y almacena el nodo raíz del árbol HTML analizado.
- **Clase Content**: Proporciona funciones útiles para la tokenización de cadenas HTML.
- **Clases HtmlNode y TextNode**: Representan los elementos y nodos de texto en un documento HTML.
- **Clase Tag**: Representa las etiquetas HTML y sus atributos.

Análisis detallado de los pasos

1. Inicialización La clase `Dom` es responsable de inicializar el proceso de análisis. El método `loadStr` toma una cadena de HTML cruda, la limpia e inicializa un objeto `Content`.

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

2. Tokenización La clase `Content` proporciona funciones utilitarias para tokenizar cadenas HTML. Incluye métodos para copiar caracteres desde la posición actual del carácter, omitir caracteres y manejar tokens como etiquetas y atributos.

- **copyUntil**Copia caracteres desde la posición actual hasta encontrar el carácter especificado.
- **skipByToken**Salta caracteres según el token especificado.

Estos métodos se utilizan para identificar y extraer diferentes partes de un HTML, como etiquetas, atributos y contenido de texto.

3. Construcción de la estructura de árbol El método `parse` de la clase `Dom` recorre la cadena de HTML, identifica las etiquetas y el texto, y construye una estructura de árbol compuesta por `HtmlNode` y `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)
        }
    }
}
```

- **Nodo raíz**: El análisis comienza desde el nodo raíz (`HtmlNode`, con la etiqueta “root”).
- **Nodo activo**: La variable `activeNode` rastrea el nodo que se está procesando actualmente.
- **Contenido de texto**: Si se detecta contenido de texto, se crea un `TextNode` y se añade al nodo actual.
- **Análisis de etiquetas**: Si se detecta una etiqueta, se llama al método `parseTag` para procesarla.

Análisis de Etiquetas El método `parseTag` se encarga de la identificación y procesamiento de las etiquetas.

```
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
}
```

- **Identificación de etiquetas**: Este método identifica si una etiqueta es de apertura o de cierre.
- **Atributos**: Analiza los atributos de la etiqueta y los añade al `HtmlNode`.
- **Etiquetas auto-cerradas**: Maneja adecuadamente las etiquetas que se cierran automáticamente.

Conclusión

El proceso de análisis de Creak implica la inicialización del contenido HTML, su tokenización y la construcción de una estructura de árbol de nodos. La clase `Dom` gestiona el análisis general, mientras que la clase `Content` proporciona funciones útiles para tokenizar cadenas de HTML. Las clases `HtmlNode` y `TextNode` representan elementos y texto en un documento HTML, y la clase `Tag` gestiona los atributos de las etiquetas. Este enfoque eficiente y organizado convierte a Creak en una herramienta poderosa para analizar HTML en Swift.