Creak: □□ Swift □ HTML □□□

Creak ist darauf ausgelegt, HTML-Dokumente effizient zu parsen und eine Baumstruktur zu erstellen, die die Elemente des Dokuments repräsentiert. Der Parsing-Prozess umfasst mehrere Schlüsselschritte und Komponenten, die zusammenarbeiten, um dieses Ziel zu erreichen. Im Folgenden finden Sie eine detaillierte Erläuterung, wie Creak HTML parst:

Überblick über den Analyseprozess

1. **Initialisierung**: Laden und Bereinigen des HTML-Strings.
2. **Tokenisierung**: Zerlegen des HTML-Strings in Tokens, die verschiedene Teile des HTML repräsentieren, wie Tags und Text.
3. **Baumstruktur aufbauen**: Verwenden der Tokens, um eine Baumstruktur zu erstellen, die die Elemente und Texte des HTML-Dokuments darstellt.

Schlüsselkomponenten

- **Dom-Klasse**: Verwaltet den gesamten Parsing-Prozess und speichert den Wurzelknoten des geparsten HTML-Baums.
- **Content-Klasse**: Stellt praktische Funktionen zur Tokenisierung von HTML-Strings bereit.
- **HtmlNode- und TextNode-Klassen**: Repräsentieren Elemente und Textknoten in einem HTML-Dokument.
- **Tag-Klasse**: Repräsentiert HTML-Tags und deren Attribute.

Detaillierte Schritt-für-Schritt-Erklärung

1. Initialisierung Die `Dom`-Klasse ist für die Initialisierung des Parsing-Prozesses verantwortlich. Die Methode `loadStr` nimmt den rohen HTML-String entgegen, bereinigt ihn und initialisiert das `Content`-Objekt.

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

```
}
```

Übersetzung:

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

Hinweis: Der Code wurde nicht übersetzt, da es sich um einen technischen Codeblock handelt, der in der Regel in der Originalsprache belassen wird, um die Integrität und Funktionalität des Codes zu gewährleisten.

2. Tokenisierung Die `Content`-Klasse bietet Hilfsfunktionen zur Tokenisierung von HTML-Strings. Sie enthält Methoden zum Kopieren von Zeichen ab der aktuellen Position, zum Überspringen von Zeichen sowie zur Verarbeitung von Tokens wie Tags und Attributen.

- **copyUntil**Kopiert Zeichen von der aktuellen Position, bis ein bestimmtes Zeichen gefunden wird.
- **skipByToken**Überspringt Zeichen basierend auf einem angegebenen Token.

Diese Methoden dienen dazu, verschiedene Teile von HTML zu identifizieren und zu extrahieren, wie zum Beispiel Tags, Attribute und Textinhalte.

3. Baumstruktur aufbauen Die `parse`-Methode in der `Dom`-Klasse durchläuft den HTML-String, erkennt Tags und Texte und baut eine Baumstruktur aus `HtmlNode`- und `TextNode`-Objekten auf.

```
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)
        }
    }
}
```

- **Wurzelknoten**: Die Analyse beginnt beim Wurzelknoten (`HtmlNode`, Tag “root”).
- **Aktiver Knoten**: Die Variable `activeNode` verfolgt den aktuell bearbeiteten Knoten.
- **Textinhalt**: Wenn Textinhalt gefunden wird, wird ein `TextNode` erstellt und dem aktuellen Knoten hinzugefügt.
- **Tag-Analyse**: Wenn ein Tag gefunden wird, wird die Methode `parseTag` aufgerufen, um es zu verarbeiten.

Tag-Analyse Die Methode `parseTag` ist für die Erkennung und Verarbeitung von Tags zuständig.

```
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-Erkennung**: Diese Methode erkennt, ob ein Tag ein öffnendes oder ein schließendes Tag ist.
- **Attribute**: Die Attribute des Tags werden analysiert und dem `HtmlNode` hinzugefügt.
- **Selbstschließende Tags**: Selbstschließende Tags werden korrekt behandelt.

Fazit

Der Parsing-Prozess von Creak umfasst die Initialisierung von HTML-Inhalten, deren Tokenisierung und den Aufbau einer Baumstruktur von Knoten. Die Klasse `Dom` verwaltet den gesamten Parsing-Prozess, während die Klasse `Content` praktische Funktionen für die Tokenisierung von HTML-Strings bereitstellt. Die Klassen `HtmlNode` und `TextNode` repräsentieren Elemente und Texte innerhalb eines HTML-Dokuments, und die Klasse `Tag` verwaltet die Attribute von Tags. Dieser effiziente und organisierte Ansatz macht Creak zu einem leistungsstarken Werkzeug für das Parsing von HTML in Swift.