Creak: Swift の HTML 解析ライブラリ

Creak は、HTML ドキュメントを効率的に解析し、ドキュメントの要素を表すツリー構造を構築するために設計されています。解析プロセスには、この目標を達成するために協力して動作する複数の重要なステップとコンポーネントが含まれています。以下に、Creak が HTML を解析する詳細な説明を示します：

解析プロセスの概要

1. **初期化**：HTML 文字列を読み込み、クリーンアップします。
2. **トークン化**：HTML 文字列を、タグやテキストなど、HTML の異なる部分を表すトークンに分解します。
3. **ツリー構造の構築**：トークンを使用して、HTML ドキュメントの要素とテキストを表すツリー構造を構築します。

主要コンポーネント

- **Dom クラス**：解析プロセス全体を管理し、解析された HTML ツリーのルートノードを保存します。
- **Content クラス**：HTML 文字列をトークン化するためのユーティリティ関数を提供します。
- **HtmlNode クラスと TextNode クラス**：HTML ドキュメント内の要素とテキストノードを表します。
- **Tag クラス**：HTML タグとその属性を表します。

詳細な解析手順

1. 初期化 `Dom` クラスは、解析プロセスの初期化を担当します。`loadStr` メソッドは、生の HTML 文字列を受け取り、それをクリーンアップして `Content` オブジェクトを初期化します。

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

上記の Swift コードは、文字列を読み込んで DOM（Document Object Model）を生成する関数です。この関数は、与えられた文字列をクリーンアップし、その内容を解析して DOM を構築します。最終的に、自身のインスタンスを返します。

2. トークン化 `Content` クラスは、HTML 文字列をトークン化するためのユーティリティ関数を提供します。これには、現在の文字位置から文字をコピーする、文字をスキップする、タグや属性などのトークンを処理するメソッドが含まれます。

- **copyUntil**：現在の位置から指定された文字に遭遇するまで文字をコピーします。
- **skipByToken**：指定されたトークンに基づいて文字をスキップします。

これらのメソッドは、HTML のさまざまな部分（例えば、タグ、属性、テキストコンテンツなど）を識別し、抽出するために使用されます。

3. ツリー構造の構築 `Dom` クラスの `parse` メソッドは、HTML 文字列を走査し、タグとテキストを識別して、`HtmlNode` と `TextNode` で構成されるツリー構造を構築します。

```
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)
    }
  }
}
```

この Swift コードは、HTML コンテンツを解析するための `parse` 関数を定義しています。以下にその動作を説明します。

1. **初期化**: `root` ノードを作成し、`activeNode` を `root` に設定します。
2. **ループ**: `activeNode` が `nil` でない限り、ループを続けます。
3. **文字列の取得**: `<` までの文字列を取得します。
4. **タグの解析**:
   - 文字列が空の場合、`parseTag` 関数を呼び出してタグ情報を取得します。
   - タグが閉じタグの場合、対応する開始タグを見つけるまで親ノードを遡ります。
   - タグが開始タグの場合、新しいノードを作成し、`activeNode` の子ノードとして追加します。自己終了タグでない場合、`activeNode` を新しいノードに更新します。
5. **テキストノードの追加**: 文字列が空でない場合、テキストノードを作成し、`activeNode` の子ノードとして追加します。

この関数は、HTML の構造を解析し、ノードツリーを構築するために使用されます。

- **ルートノード**: 解析はルートノード (`HtmlNode`、タグ名 “root”) から始まります。
- **アクティブノード**: `activeNode` 変数は、現在処理中のノードを追跡します。
- **テキストコンテンツ**: テキストコンテンツが見つかった場合、`TextNode` が作成され、現在のノードに追加されます。
- **タグ解析**: タグが見つかった場合、`parseTag` メソッドが呼び出されて処理されます。

タグ解析 `parseTag` メソッドは、タグの識別と処理を行います。

```
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
}
```

この Swift コードは、HTML タグを解析するための関数 `parseTag()` を定義しています。以下にその主要な部分を日本語で説明します。

1. **初期化と基本チェック**:
    - `result` は `ParseInfo` 型の変数で、解析結果を保持します。

- `content.char()` が `<` でない場合、`result` をそのまま返します。

2. **閉じタグの解析**:
   - `content.fastForward(1).char()` が`/`の場合、閉じタグであると判断します。
   - タグ名を取得し、`selfClosing` リストに含まれているかどうかをチェックします。
   - 含まれている場合は `result.status` を `true` に設定して返します。
   - 含まれていない場合は、`result` にタグ名と閉じタグであることを示すフラグを設定して返します。
3. **開始タグの解析**:
   - タグ名を取得し、`HtmlNode` オブジェクトを作成します。
   - `>` または`/`に到達するまで、属性を解析します。
   - 属性名と値を取得し、`HtmlNode` に設定します。
4. **自己閉じタグのチェック**:
   - タグが自己閉じタグであるかどうかをチェックし、`HtmlNode` に設定します。
5. **結果の返却**:
   - 解析が成功した場合、`result.status` を `true` に設定し、`result.node` に解析された `HtmlNode` を設定して返します。

この関数は、HTML タグを解析し、その結果を `ParseInfo` オブジェクトとして返す役割を果たします。

- **タグ識別**: この方法は、タグが開始タグか終了タグかを識別します。
- **属性**: タグの属性を解析し、`HtmlNode` に追加します。
- **自己終了タグ**: 自己終了タグを適切に処理します。

結論

Creak の解析プロセスは、HTML コンテンツの初期化、トークン化、そしてノードのツリー構造の構築を含みます。`Dom` クラスは全体の解析を管理し、`Content` クラスはトークン化された HTML 文字列のユーティリティ関数を提供します。`HtmlNode` と `TextNode` クラスは HTML ドキュメント内の要素とテキストを表し、`Tag` クラスはタグの属性を管理します。この効率的で組織的なアプローチにより、Creak は Swift における HTML 解析の強力なツールとなっています。