□□□□□: एक □□□□□ का □□□□ पार्सिंग लाइब्रेरी

□□□□□ को □□□□ दस्तावेज़ों को कुशलतापूर्वक पार्स करने और दस्तावेज़ तत्वों को दर्शाने वाले ट्री संरचना का निर्माण करने के लिए डिज़ाइन किया गया है। पार्सिंग प्रक्रिया में कई महत्वपूर्ण चरण और घटक शामिल होते हैं, जो इस लक्ष्य को प्राप्त करने के लिए मिलकर काम करते हैं। निम्नलिखित □□□□□ द्वारा □□□□ को पार्स करने का विस्तृत विवरण है:

विश्लेषण प्रक्रिया का सिंहावलोकन

1. **प्रारंभिकरण (□□□□□□□□□□□□□□□□)**: □□□□ स्ट्रिंग को लोड करें और साफ़ करें।
2. **टोकनाइज़ेशन (□□□□□□□□□□□□□□)**: □□□□ स्ट्रिंग को टोकन में विभाजित करें जो □□□□ के विभिन्न भागों को दर्शाते हैं, जैसे टैग और टेक्स्ट।
3. **ट्री संरचना निर्माण (□□□□ □□□□□□□□□□ □□□□□□□□□□□□□□)**: टोकन का उपयोग करके एक ट्री संरचना बनाएं जो □□□□ दस्तावेज़ के तत्वों और टेक्स्ट को दर्शाती है।

मुख्य घटक

- □ □□□ **क्लास**: पूरी पार्सिंग प्रक्रिया को प्रबंधित करता है और पार्स किए गए □□□□ ट्री के रूट नोड को संग्रहीत करता है।
- □ □□□□□□□ **क्लास**: □□□□ स्ट्रिंग को टोकनाइज़ करने के लिए उपयोगी फ़ंक्शन प्रदान करता है।
- □ □□□□□□□□ **और** □□□□□□□□ **क्लास**: □□□□ दस्तावेज़ में तत्व और टेक्स्ट नोड्स को प्रदर्शित करते हैं।
- □ □□□ **क्लास**: □□□□ टैग और उसके गुणों को प्रदर्शित करता है।

विस्तृत विश्लेषण चरण

1. प्रारंभिकरण (□□□□□□□□□□□□□□□□) `Dom` क्लास पार्सिंग प्रक्रिया को प्रारंभ करने के लिए जिम्मेदार है। `loadStr` मेथड कच्चे □□□□ स्ट्रिंग को स्वीकार करता है, उसे साफ करता है, और `Content` ऑब्जेक्ट को प्रारंभ करता है।

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

यह कोड □□□□□ में लिखा गया है और इसमें एक फ़ंक्शन `loadStr` है जो एक स्ट्रिंग `str` को लेता है और एक `Dom` ऑब्जेक्ट को लौटाता है। इस फ़ंक्शन में निम्नलिखित कदम होते हैं:

1. `raw` वेरिएबल को `str` स्ट्रिंग से सेट किया जाता है।
2. `clean` फ़ंक्शन का उपयोग करके `str` को साफ़ किया जाता है और परिणाम `html` में स्टोर किया जाता है।
3. `content` वेरिएबल को `Content` ऑब्जेक्ट के रूप में इनिशियलाइज़ किया जाता है, जिसमें `html` स्ट्रिंग पास की जाती है।
4. `parse` फ़ंक्शन को कॉल किया जाता है।
5. अंत में, `self` (यानी, वर्तमान `Dom` ऑब्जेक्ट) को लौटाया जाता है।

इस कोड को हिंदी में समझाने के लिए, यह एक स्ट्रिंग को लेता है, उसे साफ़ करता है, और फिर उसे पार्स करके एक `Dom` ऑब्जेक्ट के रूप में लौटाता है।

2. टोकनाइज़ेशन `Content` क्लास □□□□ स्ट्रिंग को टोकनाइज़ करने के लिए उपयोगी फ़ंक्शन प्रदान करता है। इसमें वर्तमान वर्ण स्थिति से वर्णों को कॉपी करना, वर्णों को छोड़ना, और टैग और गुणों जैसे टोकन को संभालने के तरीके शामिल हैं।

- □ □□□□□□□□□□□वर्तमान स्थिति से वर्णों को कॉपी करें, जब तक कि निर्दिष्ट वर्ण नहीं मिल जाता।
- □ □□□□□□□□□□□□निर्दिष्ट टोकन के आधार पर वर्णों को छोड़ें।

ये विधियाँ □□□□ के विभिन्न भागों को पहचानने और निकालने के लिए उपयोग की जाती हैं, जैसे कि टैग, गुण और पाठ सामग्री।

3. ट्री संरचना का निर्माण `Dom` क्लास में `parse` मेथड □□□□ स्ट्रिंग को पार्स करती है, टैग और टेक्स्ट को पहचानती है, और `HtmlNode` और `TextNode` से बने ट्री स्ट्रक्चर का निर्माण करती है।

```
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)
        }
    }
}
```

हिंदी व्याख्या:

यह कोड एक □□□□ पार्सर का हिस्सा है जो □□□□ कंटेंट को पार्स करता है और एक ट्री स्ट्रक्चर बनाता है। यहां `parse()` फ़ंक्शन □□□□ कंटेंट को पार्स करने के लिए उपयोग किया जाता है।

- `root` नोड को "□□□□" टैग के साथ इनिशियलाइज़ किया जाता है।
- `activeNode` वर्तमान में प्रोसेस किए जा रहे नोड को दर्शाता है।
- `content.copyUntil("<")` फ़ंक्शन का उपयोग करके □□□□ कंटेंट को < तक कॉपी किया जाता है।
- यदि कॉपी की गई स्ट्रिंग खाली है, तो `parseTag()` फ़ंक्शन का उपयोग करके टैग को पार्स किया जाता है।
- यदि टैग बंद होने वाला है (`info.closing`), तो `activeNode` को उसके पैरेंट नोड पर ले जाया जाता है।

- यदि टैग खुलने वाला है और `selfClosing` नहीं है, तो नया नोड `activeNode` के चाइल्ड के रूप में जोड़ा जाता है और `activeNode` को इस नए नोड पर सेट किया जाता है।
- यदि कॉपी की गई स्ट्रिंग खाली नहीं है और ट्रिम करने के बाद भी खाली नहीं है, तो इसे `TextNode` के रूप में जोड़ा जाता है।

यह प्रक्रिया तब तक चलती है जब तक कि सभी □□□□ कंटेंट पार्स नहीं हो जाता।

- **रूट नोड**: पार्सिंग रूट नोड (`HtmlNode`, टैग "□□□□") से शुरू होती है।
- **सक्रिय नोड**: `activeNode` वेरिएबल वर्तमान में प्रोसेस किए जा रहे नोड को ट्रैक करता है।
- **टेक्स्ट सामग्री**: यदि टेक्स्ट सामग्री पाई जाती है, तो एक `TextNode` बनाया जाता है और इसे वर्तमान नोड में जोड़ा जाता है।
- **टैग पार्सिंग**: यदि टैग पाया जाता है, तो इसे संसाधित करने के लिए `parseTag` मेथड को कॉल किया जाता है।

टैग पार्सिंग `parseTag` विधि टैग की पहचान और प्रसंस्करण को संभालती है।

```
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
}
```

- **टैग पहचान**: यह विधि यह पहचानती है कि टैग खुला टैग है या बंद टैग।
- **गुण**: टैग के गुणों को पार्स करता है और उन्हें `HtmlNode` में जोड़ता है।
- **स्व-बंद टैग**: स्व-बंद टैग को उचित तरीके से संभालता है।

निष्कर्ष

□□□□□ का पार्सिंग प्रक्रिया □□□□ सामग्री को इनिशियलाइज़ करने, उसे टोकनाइज़ करने और नोड्स के ट्री स्ट्रक्चर का निर्माण करने से संबंधित है। `Dom` क्लास समग्र पार्सिंग को प्रबंधित करती है, जबकि `Content` क्लास टोकनाइज़्ड □□□□ स्ट्रिंग्स के लिए उपयोगी फ़ंक्शन प्रदान करती है। `HtmlNode` और `TextNode` क्लास □□□□ डॉक्यूमेंट में एलिमेंट्स और टेक्स्ट को प्रस्तुत करती हैं, और `Tag` क्लास टैग्स के एट्रिब्यूट्स को प्रबंधित करती है। यह कुशल और संगठित तरीका □□□□□ को □□□□□ में □□□□ पार्स करने के लिए एक शक्तिशाली टूल बनाता है।