2.语法分析

2.1 语法分析概述

代码需要检查词法分析器输出的词法单元序列是否合法，并且以该词法单元序列生成对应的抽象语法树。

一个比较简单理解的概念就是—— **JSON解析器**

看下面这一段js代码：

```
const json = '{"result":true, "count":42}';
const obj = JSON.parse(json);

console.log(obj.count);
// Expected output: 42

console.log(obj.result);
// Expected output: true
```

这是一段来自MDN的代码，对应的内容则是：

> JSON.parse() 方法用来解析 JSON 字符串，构造由字符串描述的 JavaScript 值或对象。提供可选的 reviver 函数用以在返回之前对所得到的对象执行变换 (操作)。

所以`parse()`函数做的就是检查原本无意义的意义的字符串是否符合json规范，并转化为对应的js值或者对象。

我们语法分析器要做的也是如此，只不过有了之前词法分析的帮助，我们不需要直接面对无意义的字符串，而是可以面对带有 “字典注释” 的词法单元。

但即便如此，我们这一步的任务还是颇为艰巨，毕竟我们这一步就要补充我们这门语言的语法标准了，工作量还是有的。

而产生的抽象语法树又有什么作用呢?

按照抽象语法树，我们可以选择自己对应语法的遍历顺序，依次遍历并求解对应的值——最后的结果就是代码执行的结果了，这就是下一节——求值系统需要做的事情了。

说到如此，可能还是有些抽象，我们不妨来直观感受一下抽象语法树到底是个什么东西。

```
if (3 * 5 > 10) {
    return "Hello";
} else {
    return "Goodbye";
}
```

上述代码很简单，我们一眼就能看出其执行结果和含义，而它所产生的抽象语法树（可能的形式之一），我们可以这样子来表示：

```
{   type: "if-statement",
    condition: {
        type: "operator-expression",
        operator: ">",
        left: {
            type: "operator-expression",
            operator: "*",
            left: {
                type: "integer-literal",
                value: 3
            },
            right: {
                type: "integer-literal",
                value: 5
            }
        },
        right: {
            type: "integer-literal",
            value: 10
        }
    },
    consequence: {
        type: "return-statement",
        returnValue: {
            type: "string-literal",
            value: "Hello"
        }
    },
    alternative: {
        type: "return-statement",
        returnValue: {
            type: "string-literal",
            value: "Goodbye"
        }
    }
}
```

还是觉得抽象？我们可以来看看所对应的结构图：