知乎

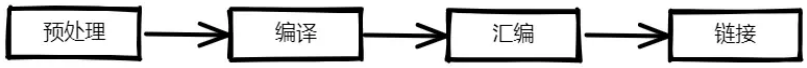
前端 ast 是什么

liubei
芸芸众生

关注

3 人赞同了该文章

在我刚开始编程时，很长一段时间都认为电脑拿到代码后直接就能运行。后来接触到编译原理后，才渐渐了解到，可执行程序是有自己的格式，我写的代码只是一串字符串，需要编译器将代码转换为可执行程序。编译器的工作有下面几个阶段



在编译阶段，编译器会经过词法分析将代码拆分成一个一个的单词，然后通过语法分析将单词转换为一个解析树。解析树里包含了所有语法信息，且能被编译器程序理解。可以认为和我们写的代码是等价的。

上边说的这几个阶段是 C 语言的编译过程，前端由于是执行脚本语言 js，相对就没有这么复杂。标题中说的 ast 叫**虚拟语法树（Abstract Syntax Tree）**，是一颗精简过的解析树。在前端开发中，通常会用到 babel 和 eslint 等工具，其内部就使用了 ast 来理解代码，识别出新语法和语法错误。前端可以用来生成 ast 的库有 acorn、babylon 等，下面用 acorn 来尝试将一段代码解析成 ast，看一下 ast 的结构。先来看下分词后的结果

```
import { Parser, tokenizer } from 'acorn';

const code = `console.log("hello, world");`;
const tokens = tokenizer(code, {});
// 输出分词后的 token 串
console.log(Array.from(tokens));
```

通过控制台可以看到，代码被分割为一个个的单词，每个单词还会有个 type 用来表示种类

```
token: ▼ (7) [Token2, Token2, Token2, Token2, Token2, Token2, Token2]
         ▶ 0: Token2 {type: TokenType2, value: 'console', start: 0, end: 7}
         ▶ 1: Token2 {type: TokenType2, value: undefined, start: 7, end: 8}
         ▶ 2: Token2 {type: TokenType2, value: 'log', start: 8, end: 11}
         ▶ 3: Token2 {type: TokenType2, value: undefined, start: 11, end: 12}
         ▶ 4: Token2 {type: TokenType2, value: 'hello, world', start: 12, end: 26}
         ▶ 5: Token2 {type: TokenType2, value: undefined, start: 26, end: 27}
         ▶ 6: Token2 {type: TokenType2, value: undefined, start: 27, end: 28}
           length: 7
         ▶ [[Prototype]]: Array(0)
```


下面看下具体生成的 ast 结构

```
import { Parser, tokenizer } from 'acorn';

code = `console.log("hello, world");`;
const ast = Parser.parse(code, {
    ecmaVersion: 2015
});
// 输出 ast
console.dir(ast);
```

```
ast: ▼ Node2 {type: 'Program', start: 0, end: 28, body: Array(1), sourceType: 'script'}
       ▼ body: Array(1)
         ▼ 0: Node2
             end: 28
           ▶ expression: Node2 {type: 'CallExpression', start: 0, end: 27, callee: Node2, arguments: Array(1)}
             start: 0
             type: "ExpressionStatement"
           ▶ [[Prototype]]: Object
           length: 1
         ▶ [[Prototype]]: Array(0)
         end: 28
         sourceType: "script"
         start: 0
         type: "Program"
       ▶ [[Prototype]]: Object
```


其可视化结构如下

知乎

ast 中的每个节点有个 type 字段，表示节点的类型。start、end 字段是节点在源码中的起始和结束位置。每种节点都有自己的附加字段，Program 类型的附加字段是 body，ExpressionStatement 类型的附加字段是 expression。这些附加字段保存了当前节点的子节点内容。之所以不能统一用 children 来标识子节点，是因为可能存在多种类型的子节点，比如下面这个 CallExpression 类型的节点，用来表示函数调用语法，用 arguments 用来标识函数实参，用 callee 标识函数名

前端对 ast 的格式已经形成了统一的规范 estree，里面规定了 js 每个版本的 ast 有多少种类型，每种类型包含了哪些字段

总结：

这篇文章只是对 ast 的一个简单的认识，下一步分析下 acorn 源码，看下 acorn 是如何进行词法分析、语法分析的。下面的仓库里有本篇文章的所有代码，以及可视化 ast 的一个小工具。

编辑于 2022-10-21 23:13

前端开发 编译原理 Babel

欢迎参与讨论

发布

知乎

还没有评论，发表第一个评论吧

推荐阅读

AST 与前端工程化实战

qiang... 发表于极乐科技

前端 DSL 实践指南（上）——内部 DSL

郑海波

前端工具链杂谈

碎碎念笔者的软件生涯主要的技术栈是 C++ 与跨平台技术（移动端），从2014年一直到2020年，都是做这方面的工作。2020年来到字节，开始做浏览器内核开发，本职工作其实还是 C++，但有一些机...

卢童鞋

【前端小白向】前端常见
盘点

写代码的海怪