



# AST 语法树



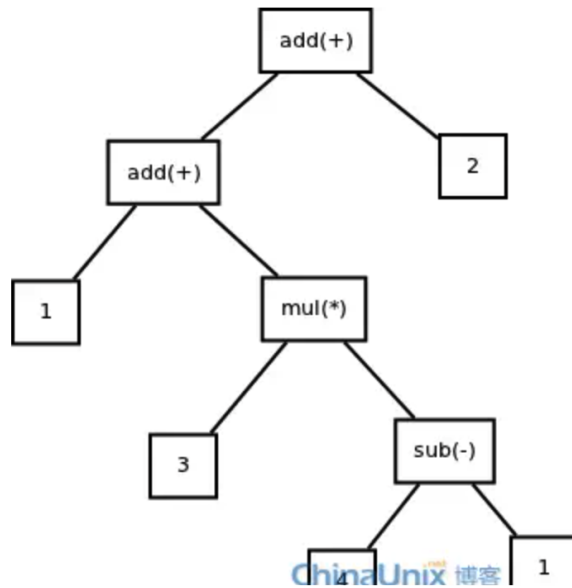
## 你将学到 Get

- 抽象语法树是什么
- 抽象语法树的应用
- Babel 的抽象语法树实战

## 抽象语法树是什么

**抽象语法树 (abstract syntax tree, AST)** 是源代码的抽象语法结构的树状表示，树上的每个节点都表示源代码中的一种结构，这所以说是抽象的，是因为抽象语法树并不会表示出真实语法出现的每一个细节，比如说，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现。抽象语法树并不依赖于源语言的语法，也就是说语法分析阶段所采用的上下文无关文法，因为在写文法时，经常会对文法进行等价的转换（消除左递归，回溯，二义性等），这样会给文法分析引入一些多余的成分，对后续阶段造成不利影响，甚至会使整个阶段变得混乱。因此，很多编译器经常要独立地构造语法分析树，为前端，后端建立一个清晰的接口。抽象语法树在很多领域有广泛的应用，比如浏览器，智能编辑器，编译器。

表达式: 1+3\*(4-1)+2



## 使用 ASTExplorer 观察 JS 代码

<https://astexplorer.net>

## Babel 的 AST 语法树的小实验

```
1 console.log(1)
2 function log(): number {
3   console.debug('before');
4   console.error(2);
5   console.debug('after');
6   return 0
7 }
8 log();
9 class Foo {
10   bar(): void {
11     console.log(3)
12   }
13   render() {
14     return ''
15   }
16 }
```

A red arrow points from the `console.error(2);` line in the code to a tooltip showing the AST node for that line:

```
function log(): number {
  console.debug('before');
  console.error("source.tsx(4,4):", 2);
  console.debug('after');
  return 0;
}
```

```
// console.log 和 console.error 中插入代码的位置信息
//一些参数的功能。
```

JavaScript

```

const parser = require('@babel/parser')
const traverse = require('@babel/traverse').default
const generator = require('@babel/generator').default
const types = require('@babel/types')
const fs = require('fs')
const fileName = 'source.tsx'

const source = fs.readFileSync(fileName).toString()

const ast = parser.parse(source, {
  plugins: ['typescript', 'jsx']
})

traverse(ast, {
  CallExpression(path) {
    const calleeStr = generator(path.node.callee).code
    console.log('calleeStr:', calleeStr)
    if (['console.log', 'console.error'].includes(calleeStr)) {
      const { line, column } = path.node.loc.start
      path.node.arguments.unshift(types.stringLiteral(`${fileName}(${
    }
  }
}))

const { code, map } = generator(ast, {
  sourceMaps: true,
  fileName
})
console.log('code ', code)

```