# ACORN.JS：原理与应用

王子龙

美团-出行事业部-技术部-前端组

美团点评

# 目录

# 关于我

- 王子龙
  - 2015.12 - 2018.4，美团-猫眼电影
  - 2018.4 - 至今，美团出行事业部
- 博客：https://borninsummer.com/

博客地址

# 内容要点

- Acorn.js
  - AST格式（ESTree Spec）
  - 遍历（walk tree）
- 工作原理
  - 词法分析
  - 语法分析
- 应用举例
  - webpack，babel
  - 项目场景1：移除无用依赖
  - 项目场景2：路径替换

# 图灵完备性与自举

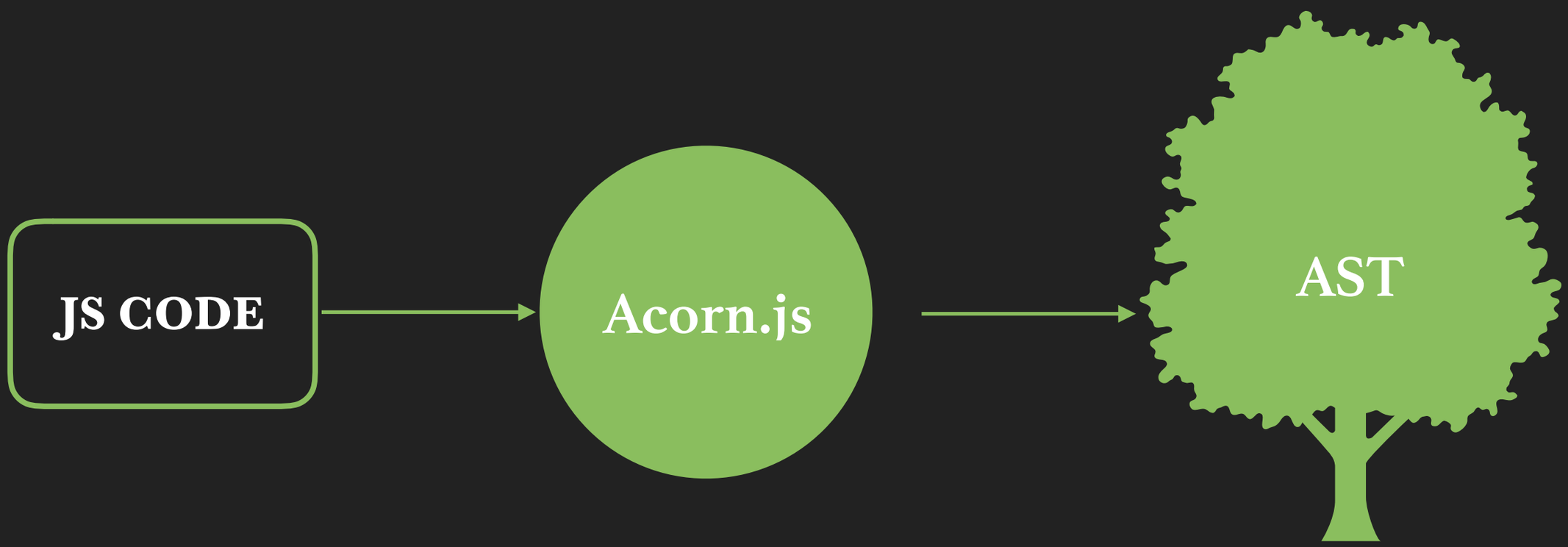
- 图灵完备（Turing Complete）
  - JS 就是一门图灵完备的语言
  - 绝大部分编程语言是图灵完备的
- 自举（Bootstrap）
  - 一门语言可以实现自己的编译器，然后可以通过编译器生成可执行代码
- 先有鸡还是先有蛋?
  - gcc

# 自举

- 通常出于推广目的考虑
- 实现了自举的编程语言，意味着该语言比较成熟了
- TypeScript 编译器实现了自举，TypeScript => JavaScript
- JavaScript 有众多的**语法分析器**实现

# ACORN.JS

A small, fast, JavaScript-based JavaScript parser. Github




作者：Marijn Haverbeke，2012 年开始开发

博客地址：https://marijnhaverbeke.nl/

# 其他类似工具

- UglifyJS：自带一个 parser
- Esprima.js：http://esprima.org/
- Babylon：Babel 从 Acorn.js fork 出来并单独演化的
- TypeScript:：已经用 TypeScript 实现了自己的 Parser
- Flow Parser：用 OCaml 实现的
- Espree，https://github.com/eslint/espree，从 Esprima fork 出来的

→ Acorn.js、Esprima.js、Babylon.js 等生成的 AST 遵从 **ESTree Spec**

# 性能对比

Run benchmarks

| | Acorn (dev) | Acorn 6.0.2 | Esprima 4.0.1 | TypeScript 3.1.3 | Traceur | Flow | Babylon |
|---|---|---|---|---|---|---|---|
| **angular.js** | 9.20 ops/sec | 10.21 ops/sec | 6.81 ops/sec | 6.92 ops/sec | | 0.56 ops/sec | 5.75 ops/sec |
| **backbone.js** | 101.78 ops/sec | 97.01 ops/sec | 84.47 ops/sec | 198.18 ops/sec | | 5.91 ops/sec | 58.13 ops/sec |
| **ember.js** | 4.57 ops/sec | 4.19 ops/sec | 3.52 ops/sec | 3.85 ops/sec | | 0.24 ops/sec | 2.01 ops/sec |
| **jquery.js** | 24.88 ops/sec | 22.78 ops/sec | 19.17 ops/sec | 28.24 ops/sec | | 1.35 ops/sec | 11.53 ops/sec |
| **react-dom.js** | 14.11 ops/sec | 13.05 ops/sec | 10.06 ops/sec | 11.28 ops/sec | | 0.72 ops/sec | 6.62 ops/sec |
| **react.js** | 65.69 ops/sec | 54.61 ops/sec | 45.98 ops/sec | 56.76 ops/sec | | 3.91 ops/sec | 36.85 ops/sec |

数据来源：https://marijnhaverbeke.nl/acorn/test/bench/index.html

# ACORN.JS-举个例子

```
1  const fs = require('fs');
2  const acorn = require('acorn');
3
4  const code = 'var a = 1;';
5  const ast = acorn.parse(code);
6
7  fs.writeFileSync('1-acorn-intro.json', JSON.stringify(ast));
```

代码示例 ①： 运行 intro 示例

# ACORN.JS 原理

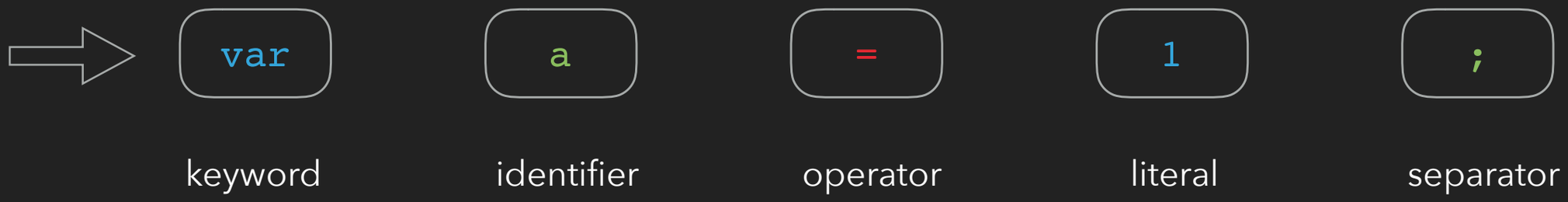
常见的编译器语法分析过程基本如下所示，Acorn.js 也是一样：

# ACORN.JS-原理-词法分析

- 词法分析：Lexical Analysis，或 lexing，或 tokenization，也叫做“扫描”（scanning）
- 过程如下：

字符流 ⇨ 词素（lexme）序列 ⇨ 词法单元（token）序列

代码： `var a = 1;`




PS：一种更常见的形式如下

```
[<keyword, var>, <id, a>, <operator, =>, <literal, 1>, <separator, ;>]
```

代码示例 ②：acorn.js 词法分析结果

# 词法分析的一些关键逻辑

- tokentype.js：定义了一些类型对象
- identifier.js：定义了 isIdentifierStart、isIdentifierChar 等函数
- 忽略空白字符（skipSpaces），包括空格、制表符、换行符
- this.pos [Integer]：遍历输入的字符流时用到的游标
- 两个 JS API：
  - String.prototype.charCodeAt()：返回0到65535之间的整数
  - String.fromCharCode()：返回 Unicode 值对应的字符

代码示例 ③：isIdentifierStart 与 isIdentifierChar 函数

# ACORN.JS-原理-语法分析

- 语法分析：Syntax Analysis，也称作“解析”（parsing）
- 语法分析的目标结果是抽象语法树（Abstract Syntax Tree，AST）
- 采用了自顶向下的方法
- 对 ECMAScript 来说，社区有一个 AST 规范：ESTree Spec
- 在线实时查看 AST：https://astexplorer.net/ 演示 => GO

# 抽象语法树的规范：ESTREE-SPEC 简介

- AST 节点按形如右侧接口定义：

```ts
interface Node {
    type: string;
    loc: SourceLocation | null;
}
```

- type 的取值分类示例：

标识符

  Identifier

字面量：

  Literal | RegExpLiteral

程序，通常只有一个，即根节点：

  Program

函数，包括函数声明和函数表达式：

  FunctionDeclaration

  FunctionExpression

语句（Statement）：

  ExpressionStatement（表达式）

  BlockStatement（块语句）

  EmptyStatement（空语句，例如单独的一个分号）

  DebuggerStatement（debugger）

  ReturnStatement

  IfStatement

  SwitchStatement

  SwitchCase

  ThrowStatement

  TryStatement

  CatchClause（catch 字句）

  WhileStatement

  DoWhileStatement

  ForStatement

声明（Declaration）

  FunctionDeclaration

  VariableDeclaration

表达式（Expression）

  FunctionExpression

  AssignmentExpression

  ...

...

更多示例 => 文档

# ACORN.JS-语法分析-部分源码解读

- node.js：定义AST节点类
- state.js：定义语法分析的主类 Parser
- statement.js：定义了语句分析用到的原型方法
  - Parser.prototype.parseTopLevel：启动分析
  - Parser.prototype.parseStatement：分析语句

查看 acorn 源码 => GO

# 遍历、操作 AST

- 遍历 AST：acorn-walk
- AST => 代码：escodegen

# ACORN.JS-应用举例-1-第三方库

- Webpack，使用 acorn.js 作为自己的 Parser 的基础库
- Babel，babylon.js: forked from acorn.js

# ACORN.JS-实际项目应用-1

- 背景：
  - mc 项目从 pc 项目中拆分出来，只保留了 src/page/ucActivity/ 目录下的页面
  - 于是，src/services 目录下定义了浏览器端所有的 ajax 请求方法，大部分可以删除。删除最大好处是减少 $service 上面挂载的属性，降低运行时浏览器内存占用
- 问题：
  - 文件数量非常多，如何安全地删除无用的文件?
- 思路：
  - 静态分析，qcs.fe.mc/src/browser-service-parser.js 查看代码 => GO

# ACORN.JS-实际项目应用-2

- qbear 处理 vue entry
- ESM 中的相对路径 => 绝对路径

```
import Vue from 'vue';
import App from '${path}';
import './styles/common.css';

Vue.config.productionTip = false

new Vue({
  el: '#app',
  render: h => h(App),
});
```

会使用 src/page/**/*/index.vue 的绝对路径来替换，
然后复制到一个**临时目录**里，引发问题：

如果其他的 import 语句使用的是相对路径，该如何处理?

方法1： webpack 配置 alias

方法2： 进行静态分析，替换为绝对路径

查看代码 => qbear 的示例

# 小结

- acorn.js 这类工具是处理代码的利器
- 常用三件套：acorn, acorn-walk, escodegen
- 延伸议题
  - 在 acorn 的基础上，如何实现 webpack 这样的打包工具?
  - 如何根据 AST 生成代码（escodegen 原理）？或者，Babel 原理
  - ESLint 工作原理

# 参考资料


- https://github.com/acornjs/acorn
- 使用 Acorn 来解析 JavaScript | 掘金
- https://en.wikipedia.org/wiki/Lexical_analysis
- https://en.wikipedia.org/wiki/Parsing
- Alfred V. Aho, etc. 编译原理（第二版）. 北京: 机械工业出版社
- http://purplebamboo.github.io/2014/09/27/javascript-syntax-tree/
- https://github.com/estree/estree/blob/master/es5.md

# Q & A