JavaScript code transformation - AST

@darkyndy

TODO

Write your own JavaScript transpiling code

TODO

Write your own JavaScript transpiling code

Write linting rules that will enforce your team code conventions

TODO

Write your own JavaScript transpiling code

Write linting rules that will enforce your team code conventions

Write “code-mods” to automatically refactor your code

TODO

Write your own JavaScript transpiling code

Write linting rules that will enforce your team code conventions

Write “code-mods” to automatically refactor your code

TODO

Write your own JavaScript transpiling code

Write linting rules that will enforce your team code conventions

Write “code-mods” to automatically refactor your code

TODO

Write your own JavaScript transpiling code

Write linting rules that will enforce your team code conventions

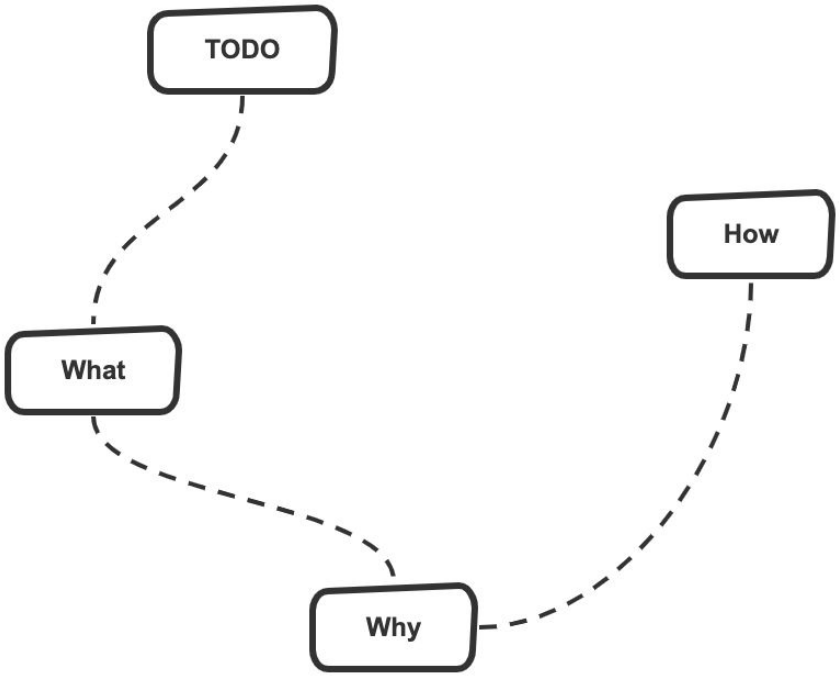
Write “code-mods” to automatically refactor your code

jscodeshift

Paul Comanici

- Technical Lead
- 10+ yrs experience working with JavaScript
- Open source supporter
- Nature lover

Journey



What

Is a tree representation of the abstract syntactic structure of source code written in a programming language.

What

example

Why

Can be edited and enhanced with information such as properties and annotations for every element it contains

Why

```
1
2 console.log('Like');
3
```

Why

```
1
2 console.log('Like');
3
```

```
- arguments:  [
   - StringLiteral = $node {
       type: "StringLiteral"
       start: 13
       end: 19
     + loc: {start, end}
     + extra: {rawValue, raw}
       value: "Like"
     }
  ]
```

Why

```
- arguments:  [
   - StringLiteral = $node {
       type: "StringLiteral"
       start: 13
       end: 19
     + loc: {start, end}
     + extra: {rawValue, raw}
       value: "Like"
     }
  ]
```

```
- arguments:  [
   - StringLiteral = $node {
       type: "StringLiteral"
       start: 13
       end: 19
     + loc: {start, end}
     + extra: {rawValue, raw}
       value: "Like"
     }
   - StringLiteral  {
       type: "StringLiteral"
       start: 21
       end: 31
     + loc: {start, end}
     + extra: {rawValue, raw}
       value: "CodeCamp"
     }
  ]
```

Why

```
- arguments:  [
   - StringLiteral = $node {
        type: "StringLiteral"
        start: 13
        end: 19
      + loc: {start, end}
      + extra: {rawValue, raw}
        value: "Like"
     }
   - StringLiteral  {
        type: "StringLiteral"
        start: 21
        end: 31
     + loc: {start, end}
     + extra: {rawValue, raw}
        value: "CodeCamp"
     }
  ]
```

```
1
2 console.log('Like', 'CodeCamp');
3
```

Why

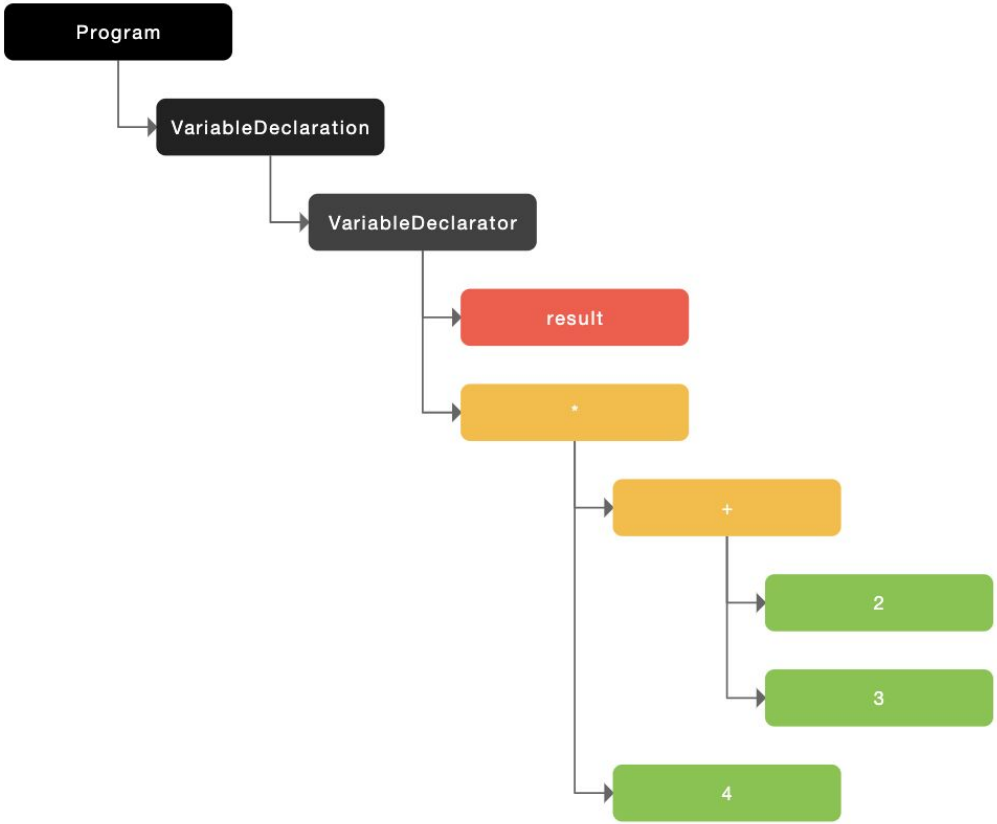
Does not include inessential information (braces, semicolons, parentheses …)

Why

```
const result = (2 + 3) * 4;
```

Why

```
const result = (2 + 3) * 4;
```



Why

Contains extra information about the program (store the position of each element in the source code)

Why

```
1 function logging() {
2   console.log('Like');
3 }
```

Why

```
1 function logging() {
2   console.log('Like');
3 }
```

```
- expression: CallExpression = $node {
    type: "CallExpression"
    start: 23
    end: 42
  + loc: {start, end}
  + callee: MemberExpression {type, start, end, loc, object, ... +2}
  - arguments:  [
    - StringLiteral  {
        type: "StringLiteral"
        start: 35
        end: 41
      + loc: {start, end}
      + extra: {rawValue, raw}
        value: "Like"
      }
    ]
  }
```

Why

- Can be edited and enhanced with information such as properties and annotations for every element it contains
- Does not include inessential information (braces, semicolons, parentheses …)
- Contains extra information about the program (store the position of each element in the source code)

How - parsers

- @babel/parser
- esprima
- acorn
- babylon

Visitor pattern

Separate algorithms from an object structure on which they operate

Visitor pattern

```
1   function Ceo() {
2     let income = 100;
3     this.setIncome = val => {
4       income = val;
5     };
6     this.getIncome = () => income;
7     this.accept = operation => {
8       operation.visit(this);
9     };
10  }
```

Visitor pattern

```
function Ceo() {
  let income = 100;
  this.setIncome = val => {
    income = val;
  };
  this.getIncome = () => income;
  this.accept = operation => {
    operation.visit(this);
  };
}
```

```
function Vp() {
  let income = 70;
  this.setIncome = val => {
    income = val;
  };
  this.getIncome = () => income;
  this.accept = operation => {
    operation.visit(this);
  };
}
```

Visitor pattern

```
21  const extraIncome = {
22    visit: position => {
23      const initialIncome = position.getIncome();
24      position.setIncome(initialIncome * 2);
25    }
26  };
```

Visitor pattern

```
31    const ceo = new Ceo();
32    ceo.accept(extraIncome);
33    console.log(ceo.getIncome()); // 200
34
35    const vp = new Vp();
36    vp.accept(extraIncome);
37    console.log(vp.getIncome()); // 140
```

Resources

- https://astexplorer.net/
- JointJS AST graph
- Babel Plugin Handbook
- JavaScript to SVG flowchart

Thank you!