拉勾教育
互联网人实战大学

《前端高手进阶》

朱德龙 前中兴软创主任工程师

— 拉勾教育出品 —

拉勾教育

拉勾教育
互联网人实战大学

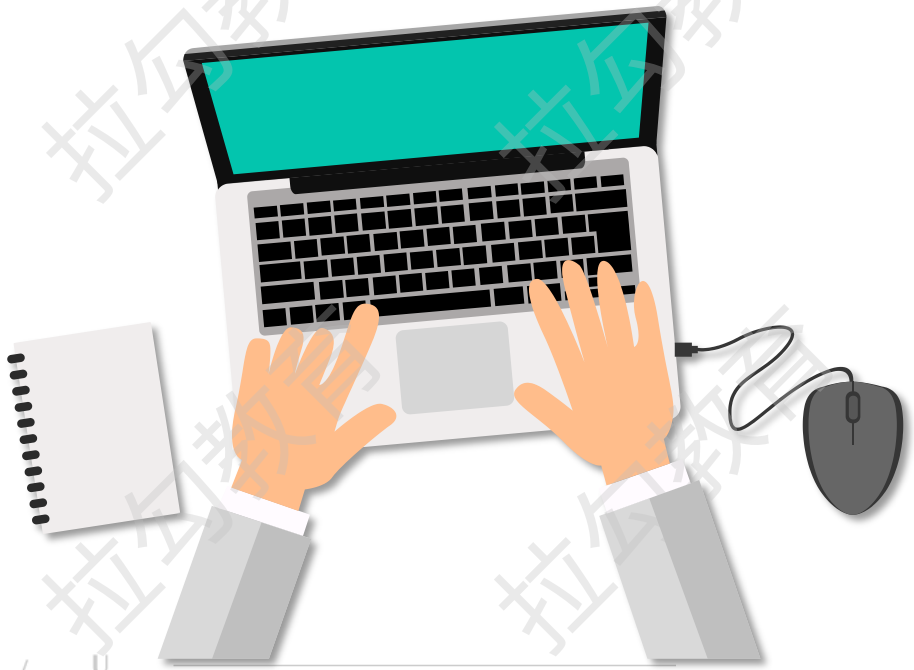
加餐01：手写 CSS 预处理

L / A / G / O / U

功能需求

CSS 预处理器的功能可以理解为精简版的 stylus

- 用空格和换行符替代花括号、冒号和分号
- 支持选择器的嵌套组合
- 支持以“$”符号开头的变量定义和使用



功能需求


```
div {color:darkkhaki;}
div p {border:1px solid lightgreen;}
div .a-b {background-color:lightyellow;}
div .a-b [data] {padding:15px;font-size:12px;}
.d-ib {display:inline-block;}
```

拉勾教育

功能需求


```
$ib inline-block
$borderColor lightgreen
div
 p
  border 1px solid $borderColor
  color darkkhaki
 .a-b
  background-color lightyellow
  [data]
   padding 15px
   font-size 12px
.d-ib
 display $ib
```

拉勾教育

编译器

拉勾教育
互联网人实战大学

对预处理器这种能将一种语言（法）转换成另一种语言（法）的程序一般称之为“**编译器**”

比如 C++、Java、JavaScript

- **解析（Parsing）**
- **转换（Transformation）**
- **代码生成（Code Generation）**

拉勾教育

编译器

解析（Parsing）

词法分析就是将接收到的源代码转换成令牌（Token）

完成这个过程的函数或工具被称之为**词法分析器（Tokenizer 或 Lexer）**

令牌由一些代码语句的碎片生成

它们可以是数字、标签、标点符号、运算符，或者其他任何东西

编译器

解析（Parsing）

将代码令牌化之后会进入语法分析

这个过程会将之前生成的令牌转换成一种带有令牌关系描述的抽象表示

这种抽象的表示称之为**抽象语法树（Abstract Syntax Tree，AST）**

完成这个过程的函数或工具被称为**语法分析器（Parser）**

编译器

转换（Transformation）

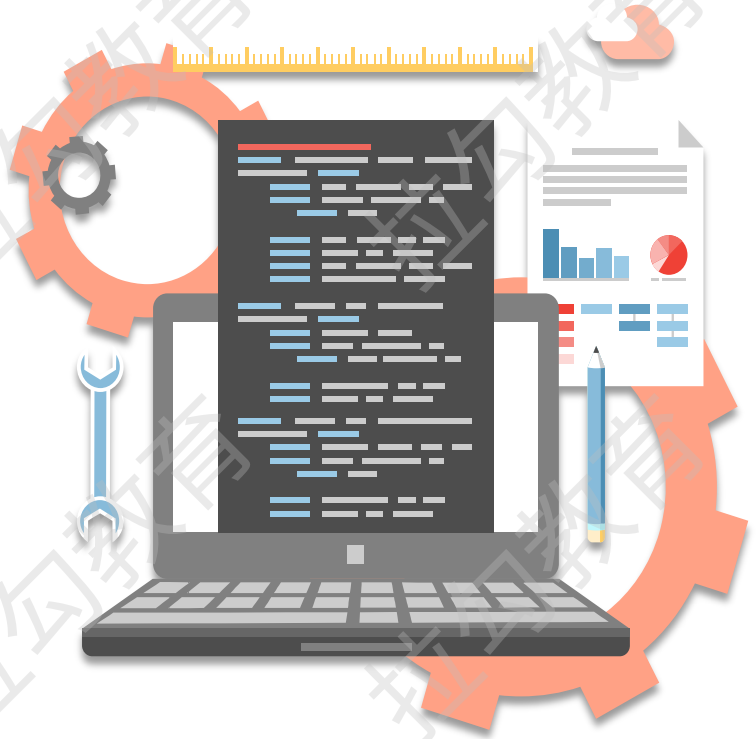
把 AST 拿过来然后做一些修改

完成这个过程的函数或工具被称之为**转换器（Transformer）**

在这个过程中，AST 中的节点可以被修改和删除

也可以新增节点

根本目的就是为了代码生成的时候更加方便



拉勾教育
互联网人实战大学

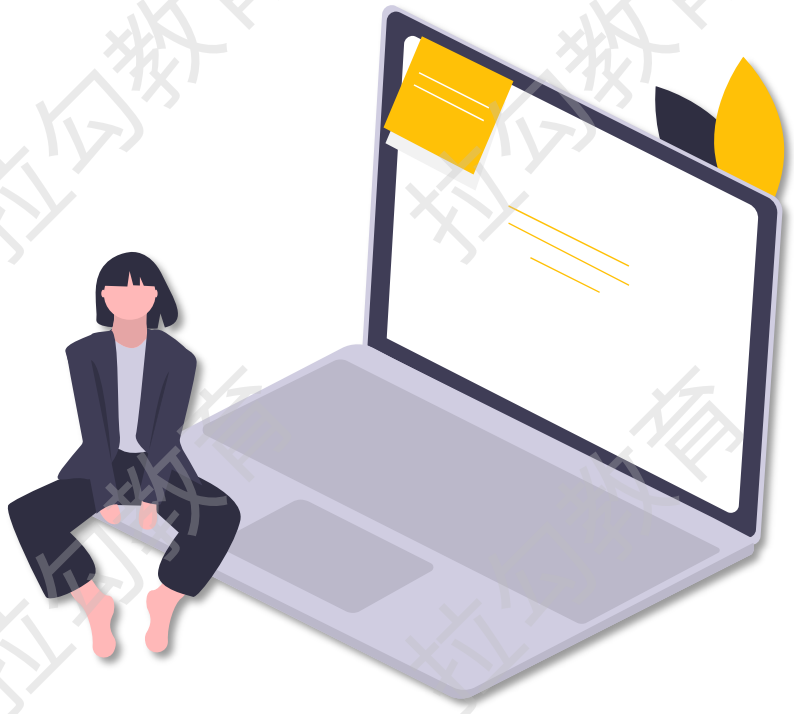
代码生成（Code Generation）

根据转换后的 AST 来生成目标代码

这个阶段做的事情有时候会和转换重叠

但是代码生成最主要的部分还是根据转换后的 AST 来输出代码

完成这个过程的函数或工具被称之为**生成器（Generator）**



拉勾教育
互联网人实战大学

代码生成（Code Generation）

代码生成有几种不同的工作方式

有些编译器将会重用之前生成的令牌

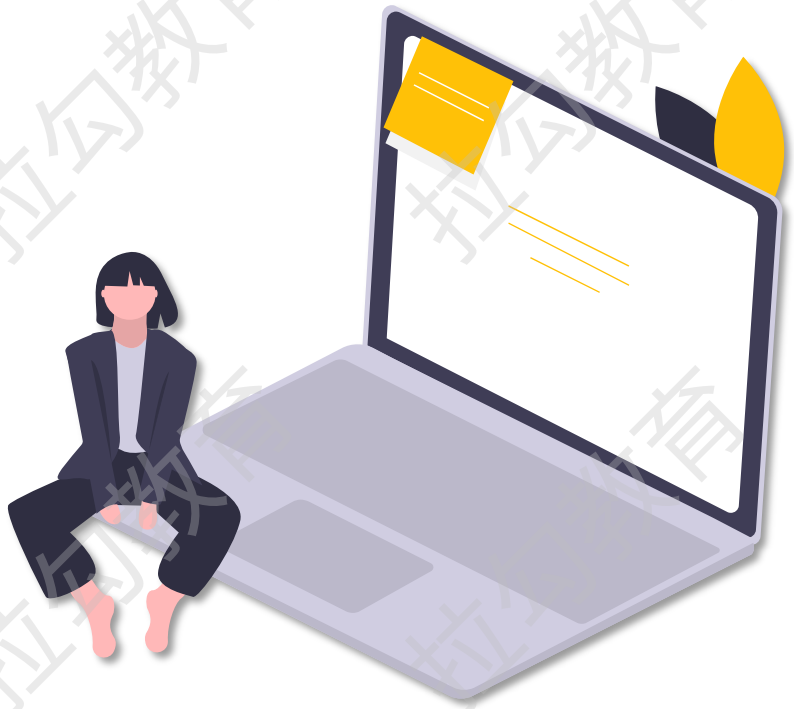
有些会创建独立代码表示，以便于线性地输出代码

但是接下来我们还是着重于使用之前生成好的 AST

代码生成器必需要知道如何“打印”转换后的 AST 中所有类型的节点

然后递归地调用自身

直到所有代码都被打印到一个很长的字符串中

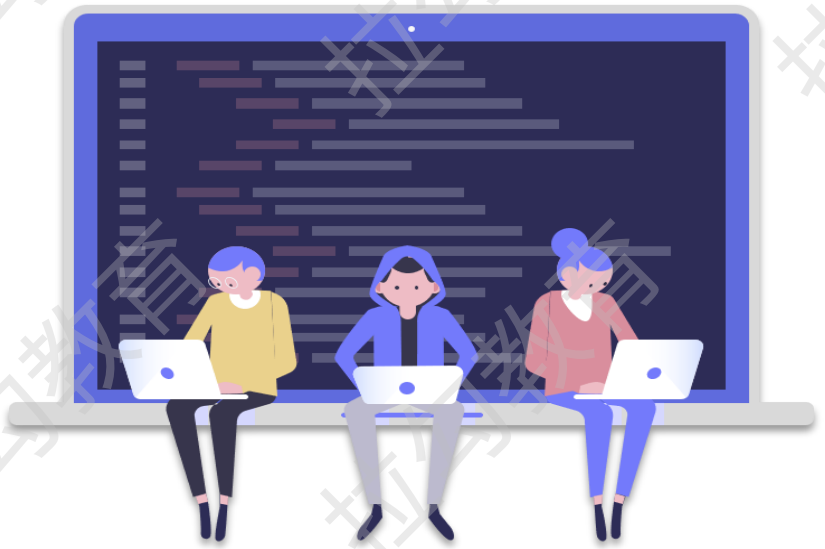


代码实现

拉勾教育
互联网人实战大学

词法分析

- 考虑字符串可以被拆分成多少种类型的令牌
- 确定令牌的判断条件及解析方式



拉勾教育

代码实现

词法分析

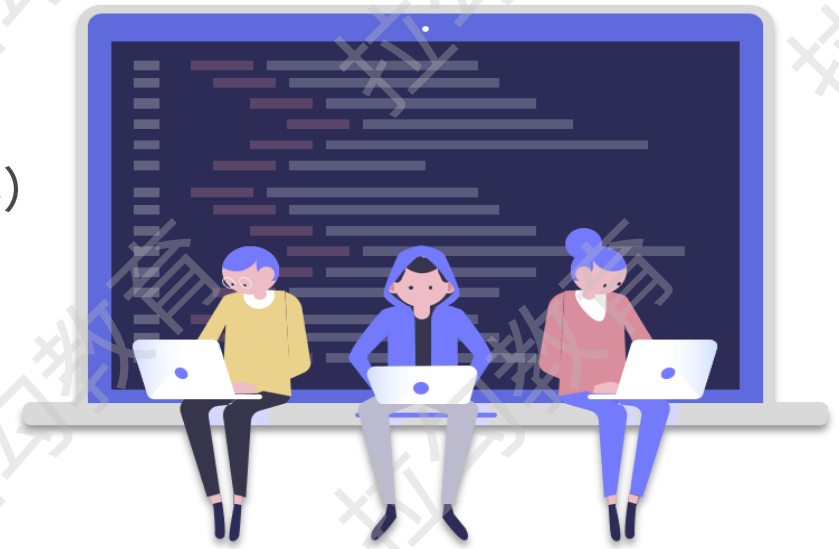
将字符串分为变量、变量值、选择器、属性、属性值 5 种类型

属性值和**变量**可以合并成一类进行处理

变量可以拆分成变量定义和变量引用

缩进会对语法分析产生影响（样式规则缩进空格数决定了属于哪个选择器）

所以也要加入令牌对象



代码实现

词法分析

- type 属性表示令牌类型
- value 属性存储令牌字符内容
- indent 属性记录缩进空格数

```
{
 type: "variableDef" | "variableRef" | "selector" | "property" |
"value", //枚举值，分别对应变量定义、变量引用、选择器、属性、值
 value: string, // token字符值，即被分解的字符串
 indent: number // 缩进空格数，需要根据它判断从属关系
}
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

词法分析

- variableDef：以“$”符号开头，该行前面无其他非空字符串
- variableRef：以“$”符号开头，该行前面有非空字符串
- selector：独占一行，该行无其他非空字符串
- property：以字母开头，该行前面无其他非空字符串
- value：非该行第一个字符串，且该行第一个字符串为 property 或 variableDef 类型

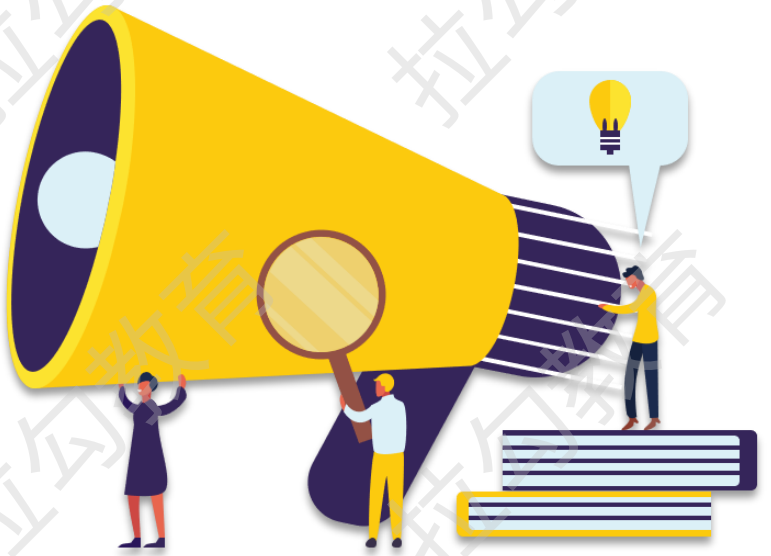
代码实现

词法分析

一般进行词法解析的时候，可以逐个字符进行解析判断

但考虑到源代码语法的特殊性——换行符和空格缩进会影响语法解析

所以可以考虑逐行**逐个单词**进行解析



拉勾教育

代码实现

词法分析

```
function tokenize(text) {
  return text.trim().split(/\n|\r\n/).reduce((tokens, line, idx) => {
    const spaces = line.match(/^\s+/) || ['']
    const indent = spaces[0].length
    const input = line.trim()
    const words = input.split(/\s/)
    let value = words.shift()
    if (words.length === 0) {
      tokens.push({
        type: 'selector',
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

词法分析

```
    type: 'selector',
    value,
    indent
  })
} else {
  let type = ''
  if (/^\$/.test(value)) {
    type = 'variableDef'
  } else if (/^[a-zA-Z-]+$/.test(value)) {
    type = 'property'
```

拉勾教育 互联网人实战大学

代码实现

词法分析

```
    type = 'property'
  } else {
    throw new Error(`Tokenize error:Line ${idx} "${value}" is not a vairable or property!`)
  }
  tokens.push({
    type,
    value,
    indent
  })
  while (value = words.shift()) {
```

拉勾教育 互联网人实战大学
L / A / G / O / U

代码实现

词法分析

```
      tokens.push({
        type: /^\$/.test(value) ? 'variableRef' : 'value',
        value,
        indent: 0
      })
    }
  }
  return tokens;
}, [])
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

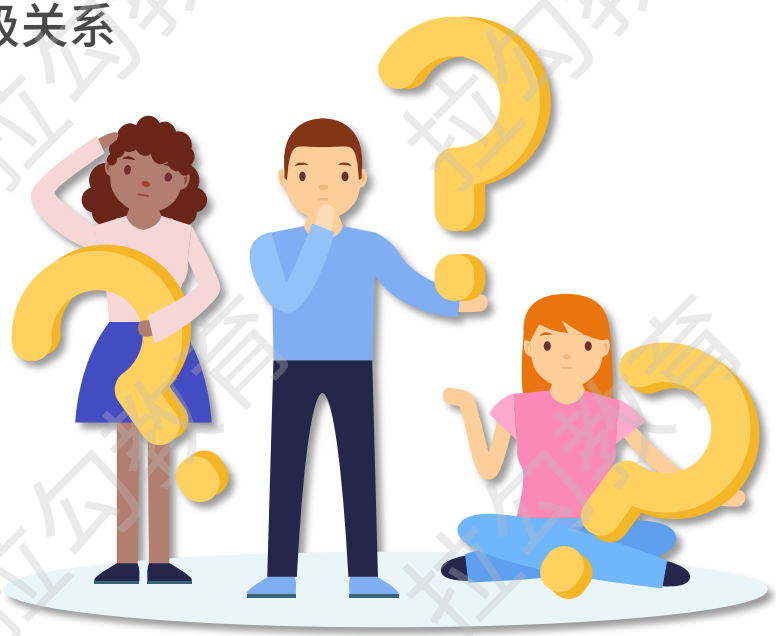
代码实现

语法分析

哪些令牌具有层级关系呢 ？

从缩进中不难看出，选择器与选择器、选择器与属性都存在层级关系

那么我们可以分别通过 children 属性和 rules 属性来描述这两类层级关系



拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

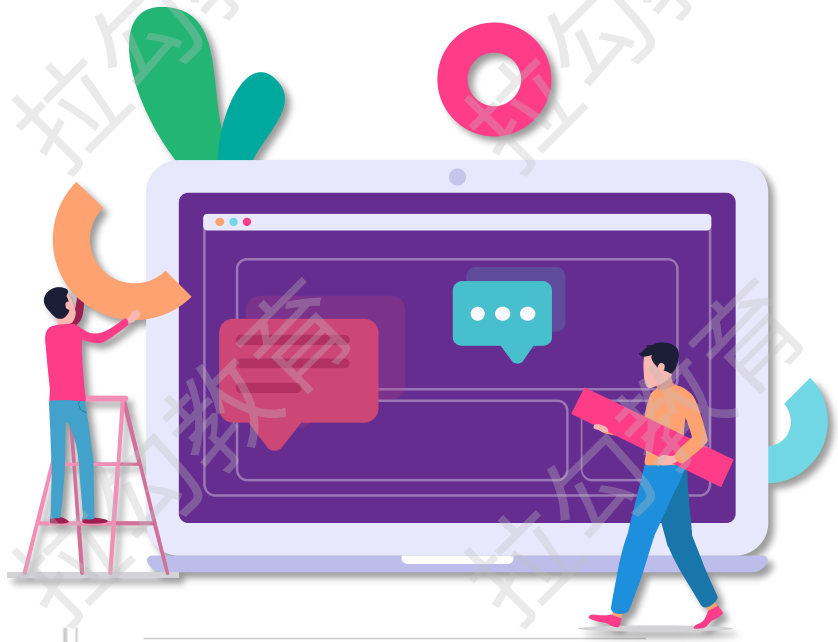
代码实现

语法分析

- 要判断层级关系需要借助缩进空格数，所以节点需要增加一个属性 indent

- 考虑到构建树时可能会产生回溯
  那么可以设置一个**数组**来记录当前构建路径
  当遇到非父子关系的节点时，沿着当前路径往上找到其父节点

- 最后为了简化树结构
  这一步也可以将变量值进行**替换**，从而减少变量节点



代码实现

语法分析

```
{
 type: 'root',
 children: [{
  type: 'selector',
  value: string
  rules: [{
    property: string,
    value: string[],
  }],
  indent: number,
  children: []
 }]
}
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

语法分析

```
function parse(tokens) {
  var ast = {
    type: 'root',
    children: [],
    indent: -1
  };
  let path = [ast]
  let preNode = ast
  let node
  let vDict = {}
  while (node = tokens.shift()) {
```

拉勾教育 互联网人实战大学

代码实现

语法分析

```
while (node = tokens.shift()) {
  if (node.type === 'variableDef') {
    if (tokens[0] && tokens[0].type === 'value') {
      const vNode = tokens.shift()
      vDict[node.value] = vNode.value
    }else {
      preNode.rules[preNode.rules.length - 1].value = vDict[node.value]
    }
    continue;
  }
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

语法分析

```
if (node.type === 'property') {
  if (node.indent > preNode.indent) {
    preNode.rules.push({
      property: node.value,
      value: []
    })
  } else {
    let parent = path.pop()
    while (node.indent <= parent.indent) {
      parent = path.pop()
    }
```

拉勾教育 互联网人实战大学

代码实现

语法分析

```
    parent.rules.push({
      property: node.value,
      value: []
    })
    preNode = parent
    path.push(parent)
  }
  continue;
}
if (node.type === 'value') {
  try {
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

语法分析

```
      preNode.rules[preNode.rules.length - 1].value.push(node.value);
    } catch (e) {
      console.error(preNode)
    }
    continue;
  }
  if (node.type === 'variableRef') {
    preNode.rules[preNode.rules.length -
1].value.push(vDict[node.value]);
    continue;
  }
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

语法分析

```
if (node.type === 'selector') {
  const item = {
    type: 'selector',
    value: node.value,
    indent: node.indent,
    rules: [],
    children: []
  }
  if (node.indent > preNode.indent) {
    path[path.length - 1].indent === node.indent && path.pop()
    path.push(item)
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

语法分析

```
    preNode.children.push(item);
    preNode = item;
  } else {
    let parent = path.pop()
    while (node.indent <= parent.indent) {
      parent = path.pop()
    }
    parent.children.push(item)
    path.push(item)
  }
}
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

语法分析

```
      let parent = path.pop()
      while (node.indent <= parent.indent) {
        parent = path.pop()
      }
      parent.children.push(item)
      path.push(item)
    }
  }
  return ast;
}
```

拉勾教育 互联网人实战大学

代码实现

转换

```
{
  selector: string,
  rules: {
    property: string,
    value: string
  }[]
}[]
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

转换

```
function transform(ast) {
  let newAst = [];
  function traverse(node, result, prefix) {
    let selector = ''
    if (node.type === 'selector') {
      selector = [...prefix, node.value];
      result.push({
        selector: selector.join(' '),
        rules: node.rules.reduce((acc, rule) => {
          acc.push({
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

转换

```
        property: rule.property,
        value: rule.value.join(' ')
      })
      return acc;
    }, [])
  })
}
for (let i = 0; i < node.children.length; i++) {
  traverse(node.children[i], result, selector)
}
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

转换

```
      }, [])
    })
  }
  for (let i = 0; i < node.children.length; i++) {
    traverse(node.children[i], result, selector)
  }
 }
 traverse(ast, newAst, [])
 return newAst;
}
```

拉勾教育
互联网人实战大学
拉勾教育
L / A / G / O / U

代码实现

代码生成

```
function generate(nodes) {
  return nodes.map(n => {
    let rules = n.rules.reduce((acc, item) => acc += `${item.property}:${item.value};`, '')
    return `${n.selector} {${rules}}`
  }).join('\n')
}
```

拉勾教育 互联网人实战大学
拉勾教育
L / A / G / O / U

总结

动手实践了一个简单的 CSS 预处理器

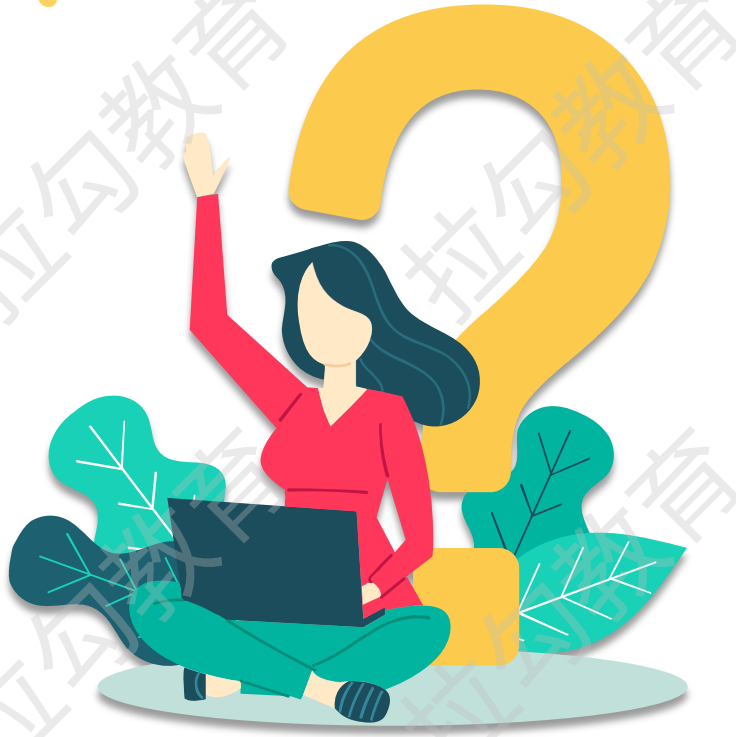
希望你能更好地掌握 CSS 工具预处理器的基本原理

同时也希望通过这个实现过程带你跨入编译器的大门

https://github.com/yalishizhude/course/blob/master/plus1/pre.js

总结

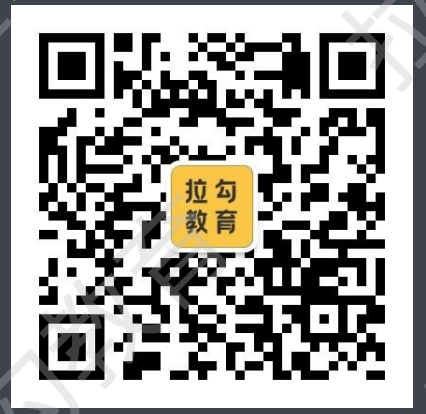
你能否为预处理器添加一些其他功能呢（比如局部变量）？



Next：第07讲《关于 JavaScript 的数据类型，你知多少？》

拉勾教育

—互联网人实战大学—



下载「拉勾教育App」
获取更多内容

L / A / G / O / U

拉勾教育