link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=39 sentences=147, words=1092

1.初始化项目

```
mkdir zhufeng-jsx-transformer
cd zhufeng-jsx-transformer
yarn add @babel/core @babel/plugin-syntax-jsx @babel/plugin-transform-react-jsx @babel/types  --dev
yarn add react
```

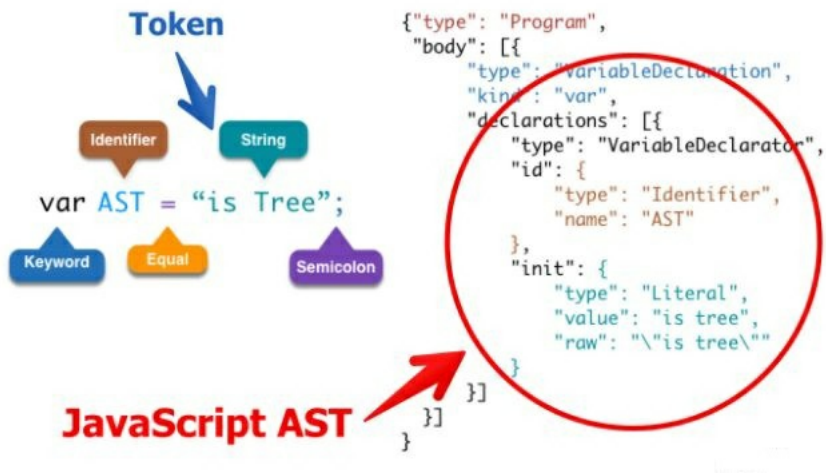
2.JSX

- React使用JSX来描述用户界面
- JSX是一种JavaScript的语法扩展
- repl (https://babeljs.io/repl)可以在线转换代码
- astexplorer (https://astexplorer.net/)可以把代码转换成AST树

```
hello
```

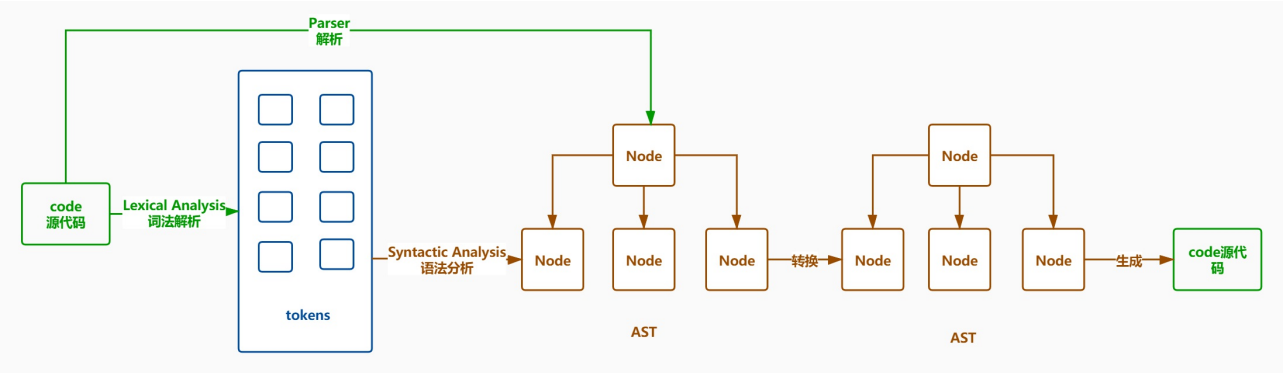
3. AST抽象语法树

- 抽象语法树（Abstract Syntax Tree，AST）是源代码语法结构的一种抽象表示。它以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的一种结构



3.1 babel工作流

- Parse(解析) 将源代码转换成抽象语法树，树上有很多的节点
- Transform(转换) 对抽象语法树进行转换
- Generate(代码生成) 将上一步经过转换过的抽象语法树生成新的代码



3.2 babel处理语法树

- babeljs (https://babeljs.io)是一个JavaScript编译器
- @babel/core (https://babeljs.io/docs/en/babel-core)是Babel编译器的核心
- babylon (https://www.npmjs.com/package/babylon)是Babel (https://github.com/babel/babel)使用的JavaScript解析器
- @babel/types (https://babeljs.io/docs/en/babel-types) 用于 AST 节点的 Lodash 式工具库
- @babel/traverse (https://babeljs.io/docs/en/babel-traverse)用于对 AST 的遍历，维护了整棵树的状态，并且负责替换、移除和添加节点
- @babel/generator (https://babeljs.io/docs/en/babel-generator)把AST转成代码

```
let babel = require('@babel/core');
let types = require('@babel/types');
let traverse = require("@babel/traverse").default;
let generate = require("@babel/generator").default;
const code = `function ast() {}`;
const ast = babel.parse(code);
let indent = 0;
const padding = ()=>" ".repeat(indent);
traverse(ast, {
    enter(path){
        console.log(padding()+path.node.type+'进入');
        indent+=2;
        if(types.isFunctionDeclaration(path.node)){
            path.node.id.name = 'newAst';
        }
    },
    exit(path){
        indent-=2;
        console.log(padding()+path.node.type+'离开');
    }
});
const output = generate(ast,{},code);
console.log(output.code);
```

3.2 旧转换

3.2.1 jsx.js

```
const babel = require("@babel/core");
const sourceCode = `hello`;
const result = babel.transform(sourceCode, {
    plugins: [['@babel/plugin-transform-react-jsx',{runtime:'classic'}]]
});
console.log(result.code);
```

3.2.2 转译结果

```
let React = require('react');
React.createElement("h1", {
  id: "title",
  key: "title",
  ref: "title"
}, "hello");
console.log(JSON.stringify(element,replacer,2));
function replacer(key,value){
    if(!['_owner','_store'].includes(key))
        return value;
}
```

```
{
  "type": "h1",
  "key": "title",
  "ref": "title",
  "props": {
    "id": "title",
    "children": "hello"
  }
}
```

3.3 新转换

3.3.1 jsx.js

```
const babel = require("@babel/core");
const sourceCode = `hello`;
const result = babel.transform(sourceCode, {
+   plugins: [['@babel/plugin-transform-react-jsx',{runtime:'automatic'}]]
});
console.log(result.code);
```

3.3.2 转译结果

```
let {jsx:_jsx} = require("react/jsx-runtime");

let element = _jsx("h1", {id: "title",key:"title",ref:"title",children: "hello"}, "title");
console.log(JSON.stringify(element,replacer,2));
function replacer(key,value){
    if(!['_owner','_store'].includes(key))
        return value;
}
```

```
{
    "type": "h1",
    "key": "title",
    "ref": "title",
    "props": {
      "id": "title",
      "children": "hello"
    }
}
```

4.实现插件

- babel-types文档 (https://babeljs.io/docs/en/babel-types.html)
- babel插件开发指南 (https://github.com/brigand/babel-plugin-handbook/blob/master/translations/zh-Hans/README.md)
- @babel/plugin-syntax-jsx (https://babeljs.io/docs/en/babel-plugin-syntax-jsx/)允许解析JSX
- babel-traverse (https://github.com/jamiebuilds/babel-handbook/blob/master/translations/en/plugin-handbook.md#babel-traverse)可用于遍历语法树
- Visitors(访问者)是一个用于 AST 遍历的模式,用于定义某个节点的操作方法
- Paths(路径) 是一个对象，它表示两个节点之间的连接
    - replaceWith可以用于替换节点
    - get用于查找特定类型的子路径
    - find 用于向上查找一个指定条件的路径
    - unshiftContainer用于把AST节点插入到类似于body这样的数组中

- Scope(作用域)
    - generateUidIdentifier会生成一个不会和任何本地定义的变量冲突的标识符

4.1 jsx.js

```
const babel = require("@babel/core");
const pluginTransformReactJsx = require('./plugin-transform-react-jsx');
const sourceCode = `hello`;
const result = babel.transform(sourceCode, {
    plugins: [pluginTransformReactJsx]
});
console.log(result.code);
```

4.2 plugin-transform-react-jsx.js

```
const types = require('@babel/types');
const pluginSyntaxJsx = require('@babel/plugin-syntax-jsx').default;
const pluginTransformReactJsx = {
  inherits:pluginSyntaxJsx,
  visitor: {
      JSXElement(path) {
      let callExpression = buildJSXElementCall(path);
      path.replaceWith(callExpression);
    }
  }
}
function buildJSXElementCall(path) {
  const args = [];
  return call(path,"jsx", args);
}
function call(path,name, args) {
  const callee = types.identifier('_jsx');
  const node = types.callExpression(callee, args);
  return node;
}
module.exports = pluginTransformReactJsx;
```

5.支持属性

plugin-transform-react-jsx.js

```
const types = require('@babel/types');
const pluginSyntaxJsx = require('@babel/plugin-syntax-jsx').default;
const pluginTransformReactJsx = {
  inherits:pluginSyntaxJsx,
  visitor: {
    JSXElement(path) {
      let callExpression = buildJSXElementCall(path);
      path.replaceWith(callExpression);
    }
  }
}
function buildJSXElementCall(path) {
+ const openingPath = path.get("openingElement");
+ const {name} = openingPath.node.name;
+ const tag = types.stringLiteral(name);
+ const args = [tag];
+ let attributes = [];
+ for (const attrPath of openingPath.get("attributes")) {
+   attributes.push(attrPath.node);
+ }
+ const children = buildChildren(path.node);
+ const props = attributes.map(convertAttribute);
+ if (children.length > 0) {
+   props.push(buildChildrenProperty(children));
+ }
+ const attributesObject = types.objectExpression(props);
+ args.push(attributesObject);
  return call(path,"jsx", args);
}
+function buildChildren(node) {
+  const elements = [];
+  for (let i = 0; i < node.children.length; i++) {
+    let child = node.children[i];
+    if (types.isJSXText(child)) {
+      elements.push(types.stringLiteral(child.value));
+    }
+  }
+  return elements;
+}
+function buildChildrenProperty(children) {
+  let childrenNode;
+  if (children.length === 1) {
+    childrenNode = children[0];
+  } else if (children.length > 1) {
+    childrenNode = types.arrayExpression(children);
+  } else {
+    return undefined;
+  }
+  return types.objectProperty(types.identifier("children"), childrenNode);
+}
+function convertAttribute(node) {
+  const value = node.value;
+  node.name.type = "Identifier";
+  return types.objectProperty(node.name, value);
+}
function call(path,name, args) {
  const callee = types.identifier('_jsx');
  const node = types.callExpression(callee, args);
  return node;
}
module.exports = pluginTransformReactJsx;
```

6.引入runtime模块

plugin-transform-react-jsx.js

```
const types = require('@babel/types');
const pluginSyntaxJsx = require('@babel/plugin-syntax-jsx').default;
const pluginTransformReactJsx = {
  inherits:pluginSyntaxJsx,
  visitor: {
    JSXElement(path) {
      let callExpression = buildJSXElementCall(path);
      path.replaceWith(callExpression);
    }
  }
}
function buildJSXElementCall(path) {
  const openingPath = path.get("openingElement");
  const {name} = openingPath.node.name;
  const tag = types.stringLiteral(name);
  const args = [tag];
  let attributes = [];
  for (const attrPath of openingPath.get("attributes")) {
    attributes.push(attrPath.node);
  }
  const children = buildChildren(path.node);
  const props = attributes.map(convertAttribute);
  if (children.length > 0) {
    props.push(buildChildrenProperty(children));
  }
  const attributesObject = types.objectExpression(props);
  args.push(attributesObject);
  return call(path,"jsx", args);
}
function buildChildren(node) {
  const elements = [];
  for (let i = 0; i < node.children.length; i++) {
    let child = node.children[i];
    if (types.isJSXText(child)) {
      elements.push(types.stringLiteral(child.value));
    }
  }
  return elements;
}
function buildChildrenProperty(children) {
  let childrenNode;
  if (children.length
    childrenNode = children[0];
  } else if (children.length > 1) {
    childrenNode = types.arrayExpression(children);
  } else {
    return undefined;
  }
  return types.objectProperty(types.identifier("children"), childrenNode);
}
function convertAttribute(node) {
  const value = node.value;
  node.name.type = "Identifier";
  return types.objectProperty(node.name, value);
}
function call(path,name, args) {
+ const importedSource = 'react/jsx-runtime';
+ const callee = addImport(path,name,importedSource);
  const node = types.callExpression(callee, args);
  return node;
}
+function addImport(path,importName,importedSource){
+  const programPath = path.find(p => p.isProgram());
+  const scope = programPath.scope;
+  const localName = scope.generateUidIdentifier(importName);
+  const specifiers  = [types.importSpecifier(localName, types.identifier(importName))];
+  let statement = types.importDeclaration(specifiers, types.stringLiteral(importedSource));
+  programPath.unshiftContainer("body", [statement]);
+  return localName;
+}
module.exports = pluginTransformReactJsx;
```