

Webpack-Day5



webpack打包原理分析

接收webpack配置 进行读取

入口：从哪个文件开始分析

哪些是依赖模块

这些依赖模块的位置 也就是路径信息

内容

对内容处理，处理成浏览器正确解析的

递归处理其他依赖模块

生成chunk 代码片段

补齐函数 生成bundle文件的内容 -》 启动器函数

出口：生成资源文件的名称和位置

今日暗号：有点感动了怎么办

作业：实现模块依赖分析函数 要求代码截图

核心功能：根据入口模块开始分析依赖路径，对内容进行处理，支持es6+ To es5的语法转换，返回模块路径，依赖，和相应处理后的

webpack 在执行npx webpack进行打包后，都干了什么事情？

```
(function(modules) {  
  var installedModules = {};  
  
  function __webpack_require__(moduleId) {  
    if (installedModules[moduleId]) {  
      return installedModules[moduleId].exports;  
    }  
    var module = (installedModules[moduleId] = {  
      i: moduleId,  
      l: false,  
      exports: {}  
    });  
    modules[moduleId].call(  
      module.exports,  
      module,  
      module.exports,  
      __webpack_require__  
    );  
    module.l = true;  
    return module.exports;  
  }  
  
  return __webpack_require__((__webpack_require__.s = "./index.js"));  
})({  
  "./index.js": function(module, exports) {  
    eval(  
      '// import a from "./a";\n\nconsole.log("hello word");\n\n\n//  
sourceURL=webpack:///./index.js?'  
    ),  
    "./a.js": function(module, exports) {  
      eval(  
        '// import a from "./a";\n\nconsole.log("hello word");\n\n\n//  
sourceURL=webpack:///./index.js?'  
      ),  
      "./b.js": function(module, exports) {  
        eval(  
          '// import a from "./a";\n\nconsole.log("hello word");\n\n\n//  
sourceURL=webpack:///./index.js?'  
        );  
      }  
    }  
  });  
});
```

大概的意思就是，我们实现了一个**webpack_require** 来实现自己的模块化，把代码都缓存在 `installedModules`里，代码文件以对象传递进来，key是路径，value是包裹的代码字符串，并且代码内部的require，都被替换成了**webpack_require**

webpack的配置文件（默认的）

- 入口（入口模块的位置）
- 出口（生成bundle文件位置）

创建一个webpack

- 接收一份配置（`webpack.config.js`）
- 分析出入口模块位置
 - 读取入口模块的内容，分析内容
 - 哪些是依赖
 - 哪些是源码
 - `es6,jsx`，处理 需要编译 -> 浏览器能够执行
 - 分析其他模块
- 拿到对象数据结构
 - 模块路径
 - 处理好的内容
- 创建**bundle.js**
 - 启动器函数，来补充代码里有可能出现的`module exports require`，让浏览器能够顺利的执行

自己实现一个**bundle.js**

- 模块分析：读取入口文件，分析代码

```
const fs = require("fs");

const fenximokuai = filename => {
  const content = fs.readFileSync(filename, "utf-8");
  console.log(content);
};

fenximokuai("./index.js");
```

- 拿到文件中依赖，这里我们不推荐使用字符串截取，引入的模块名越多，就越麻烦，不灵活，这里我们推荐使用@babel/parser，这是babel7的工具，来帮助我们分析内部的语法，包括es6，返回一个ast抽象语法树

@babel/parser:<https://babeljs.io/docs/en/babel-parser>

```
//安装@babel/parser
npm install @babel/parser --save

//bundle.js
const fs = require("fs");
const parser = require("@babel/parser");

const fenximokuai = filename => {
  const content = fs.readFileSync(filename, "utf-8");

  const Ast = parser.parse(content, {
    sourceType: "module"
  });
  console.log(Ast.program.body);
};

fenximokuai("./index.js");
```

- 接下来我们就可以根据body里面的分析结果，遍历出所有的引入模块，但是比较麻烦，这里还是推荐babel推荐的一个模块@babel/traverse，来帮我们处理。

npm install @babel/traverse --save

```
const fs = require("fs");
const path = require("path");
const parser = require("@babel/parser");
const traverse = require("@babel/traverse").default;

const fenximokuai = filename => {
  const content = fs.readFileSync(filename, "utf-8");
```

```

const Ast = parser.parse(content, {
  sourceType: "module"
});

const dependencies = [];
//分析ast抽象语法树，根据需要返回对应数据，
//根据结果返回对应的模块，定义一个数组，接受一下node.source.value的值
traverse(Ast, {
  ImportDeclaration({ node }) {
    console.log(node);
    dependencies.push(node.source.value);
  }
});
console.log(dependencies);
};

fenximokuai("./index.js");

```

```

handeMacBook-Pro:webpack2 kele$ node bundle.js
{ filename: './src/index.js',
  dependencies: { './a.js': './src/a.js' },
  code: '"use strict";\n\nvar _a = _interopRequireDefault(require("./a.js"));\n\nfunction _interopRequireDefault(obj) { return obj && obj.__esModule ? obj : { "default": obj }; }\n\nconsole.log("hello kkb");' }
handeMacBook-Pro:webpack2 kele$

```

分析上图，我们要分析出信息：

- 入口文件
- 入口文件引入的模块
 - 引入路径
 - 在项目中里的路径
- 可以在浏览器里执行的代码

处理现在的路径问题：

```

//需要用到path模块
const parser = require("@babel/parser");

//修改 dependencies 为对象，保存更多的信息
const dependencies = {};

//分析出引入模块，在项目中的路径
const newfilename =
  "." + path.join(path.dirname(filename), node.source.value);

//保存在dependencies里
dependencies[node.source.value] = newfilename;

```

把代码处理成浏览器可运行的代码，需要借助@babel/core，和@babel/preset-env，把ast语法树转换成合适的代码

```

const babel = require("@babel/core");

const { code } = babel.transformFromAst(Ast, null, {
  presets: ["@babel/preset-env"]
});

```

导出所有分析出的信息：

```

return {
  filename,
  dependencies,
  code
};

```

完成代码参考：

```

const fs = require('fs');
const path = require('path');
const parser = require('@babel/parser');
const traverse = require('@babel/traverse').default;
const babel = require('@babel/core');

```

```

const moduleAnalyser = (filename) => {
  const content = fs.readFileSync(filename, 'utf-8');
  const ast = parser.parse(content, {
    sourceType: 'module'
  });
  const dependencies = {};
  traverse(ast, {
    ImportDeclaration({ node }) {
      const dirname = path.dirname(filename);
      const newFile = './' + path.join(dirname, node.source.value);
      dependencies[node.source.value] = newFile;
    }
  });
  const { code } = babel.transformFromAst(ast, null, {
    presets: ["@babel/preset-env"]
  });
  return {
    filename,
    dependencies,
    code
  }
}

const moduleInfo = moduleAnalyser('./src/index.js');
console.log(moduleInfo);

```

- 分析依赖

上一步我们已经完成了一个模块的分析，接下来我们要完成项目里所有模块的分析：

```

const fs = require('fs');
const path = require('path');
const parser = require('@babel/parser');
const traverse = require('@babel/traverse').default;
const babel = require('@babel/core');

const moduleAnalyser = (filename) => {
  const content = fs.readFileSync(filename, 'utf-8');
  const ast = parser.parse(content, {
    sourceType: 'module'
  });
  const dependencies = {};
  traverse(ast, {

```

```

    ImportDeclaration({ node }) {
      const dirname = path.dirname(filename);
      const newFile = './' + path.join(dirname, node.source.value);
      dependencies[node.source.value] = newFile;
    }
  });
  const { code } = babel.transformFromAst(ast, null, {
    presets: ["@babel/preset-env"]
  });
  return {
    filename,
    dependencies,
    code
  }
}

const makeDependenciesGraph = (entry) => {
  const entryModule = moduleAnalyser(entry);
  const graphArray = [ entryModule ];
  for(let i = 0; i < graphArray.length; i++) {
    const item = graphArray[i];
    const { dependencies } = item;
    if(dependencies) {
      for(let j in dependencies) {
        graphArray.push(
          moduleAnalyser(dependencies[j])
        );
      }
    }
  }
  const graph = {};
  graphArray.forEach(item => {
    graph[item.filename] = {
      dependencies: item.dependencies,
      code: item.code
    }
  });
  return graph;
}

const graphInfo = makeDependenciesGraph('./src/index.js');
console.log(graphInfo);

```

- 生成代码

```

const fs = require('fs');
const path = require('path');

```



```

const parser = require('@babel/parser');
const traverse = require('@babel/traverse').default;
const babel = require('@babel/core');

const moduleAnalyser = (filename) => {
  const content = fs.readFileSync(filename, 'utf-8');
  const ast = parser.parse(content, {
    sourceType: 'module'
  });
  const dependencies = {};
  traverse(ast, {
    ImportDeclaration({ node }) {
      const dirname = path.dirname(filename);
      const newFile = './' + path.join(dirname, node.source.value);
      dependencies[node.source.value] = newFile;
    }
  });
  const { code } = babel.transformFromAst(ast, null, {
    presets: ["@babel/preset-env"]
  });
  return {
    filename,
    dependencies,
    code
  }
}

const makeDependenciesGraph = (entry) => {
  const entryModule = moduleAnalyser(entry);
  const graphArray = [ entryModule ];
  for(let i = 0; i < graphArray.length; i++) {
    const item = graphArray[i];
    const { dependencies } = item;
    if(dependencies) {
      for(let j in dependencies) {
        graphArray.push(
          moduleAnalyser(dependencies[j])
        );
      }
    }
  }
  const graph = {};
  graphArray.forEach(item => {
    graph[item.filename] = {
      dependencies: item.dependencies,
      code: item.code
    }
  });
  return graph;
}

```

```
}

const generateCode = (entry) => {
  const graph = JSON.stringify(makeDependenciesGraph(entry));
  return `
    (function(graph){
      function require(module) {
        function localRequire(relativePath) {
          return require(graph[module].dependencies[relativePath]);
        }
        var exports = {};
        (function(require, exports, code){
          eval(code)
        })(localRequire, exports, graph[module].code);
        return exports;
      };
      require('${entry}')
    })(${graph});
  `;
}

const code = generateCode('./src/index.js');
console.log(code);
```