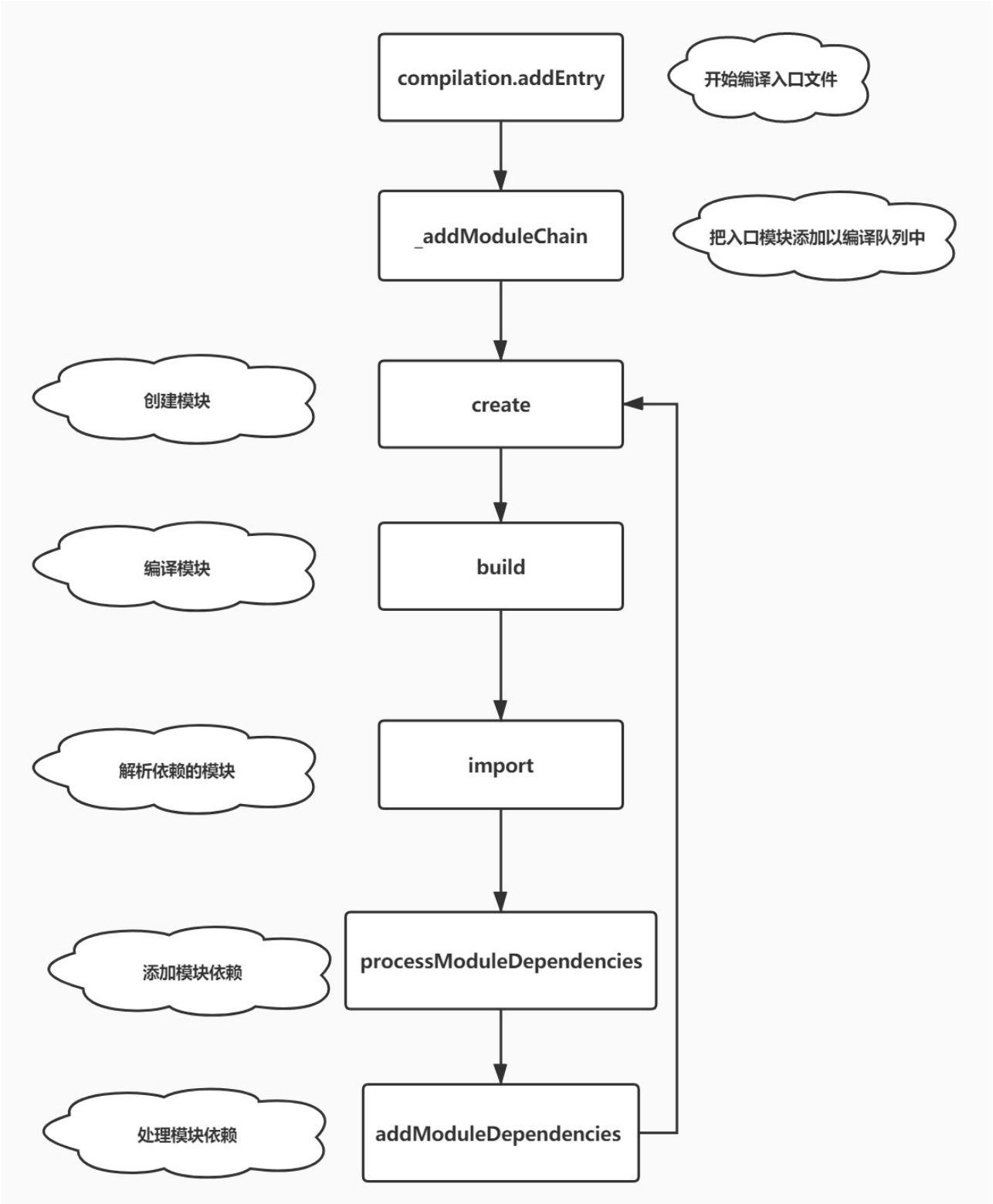
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=66 sentences=151, words=764

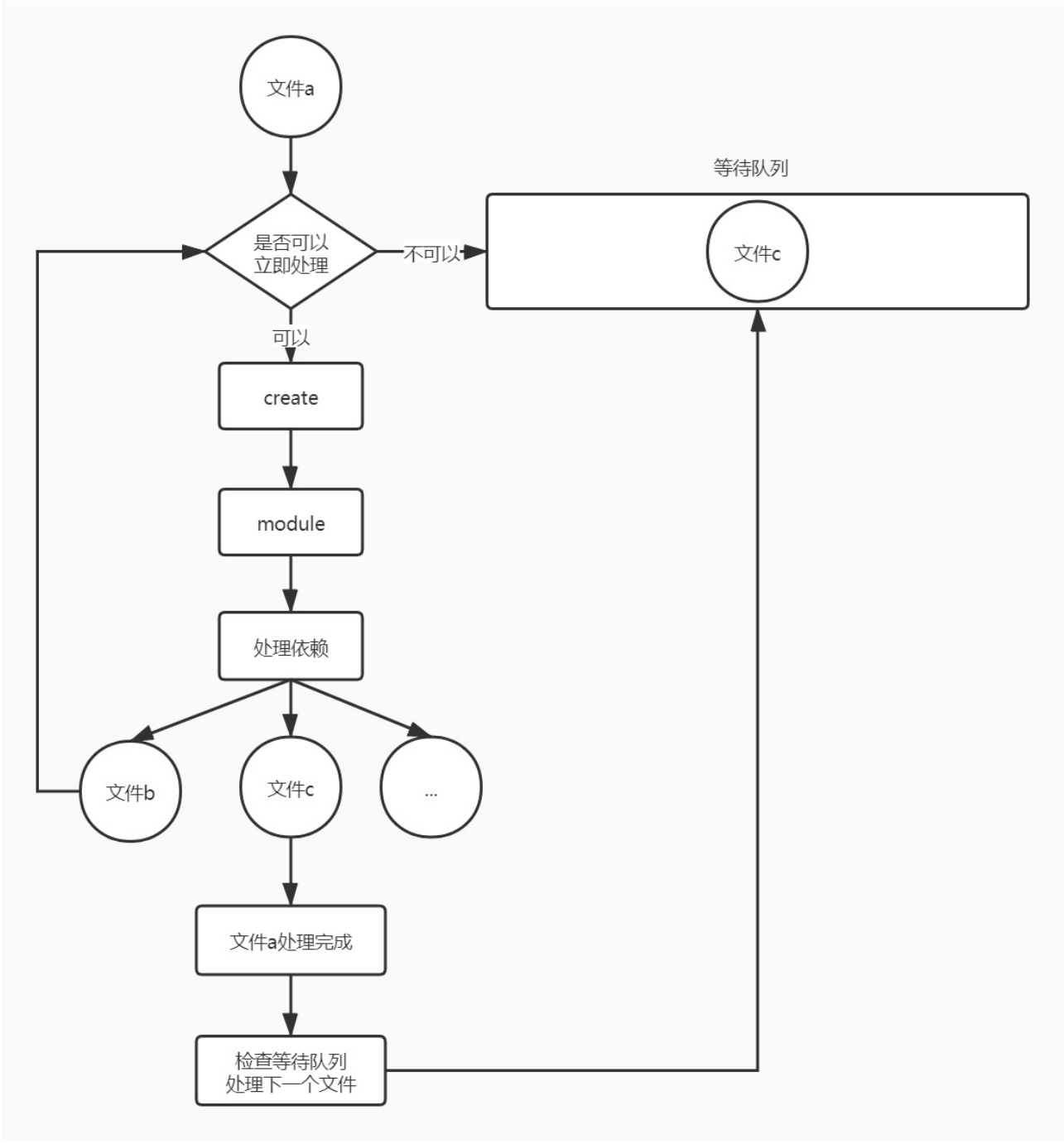
1.module

- 对于 webpack来说每个文件都是一个 module
- webpack会从配置中 entry定义开始，找到全部的文件，并转化为 module

2.build流程



3.编译队列控制



4.entry配置

4.1 字符串

```
entry: './index.js',
```

4.2 字符串数组

```
entry: ['./index1.js', './index2.js']
```

4.3 对象

```
entry: {
   main: './main.js'
}
```

4.4 函数

```
entry: () => './index.js'
entry: () => new Promise((resolve) => resolve('./index.js'))
```

5.SingleEntryPlugin

- entryOption (https://github.com/webpack/webpack/blob/v4.43.0/lib/WebpackOptionsApply.js#L290-L291)
- SingleEntryPlugin.js (https://github.com/webpack/webpack/blob/v4.43.0/lib/SingleEntryPlugin.js)

5.1 注册模块工厂

```
compiler.hooks.compilation.tap("SingleEntryPlugin",
    (compilation, { normalModuleFactory }) => {
        compilation.dependencyFactories.set(SingleEntryDependency,normalModuleFactory);
    }
);
```

5.2 make

- 注册了 make事件回调，在 make阶段的时候调用 addEntry方法，然后进入 _addModuleChain进入正式的编译阶段
- compile (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/Compiler.js#660)
- newCompilationParams (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/Compiler.js#651-L658)
- this.hooks.compilation.call (https://github.com/webpack/webpack/blob/v4.43.0/lib/Compiler.js#L631)
- this.dependencyFactories.get(Dep) (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/Compilation.js#L1054-L1063)

6.dependency

- dependencies (https://github.com/webpack/webpack/tree/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies)
- 生成 chunk时会依靠 dependency来得到依赖关系图
- 生成最终文件时会依赖 dependency中方法和保存的信息将源文件中的 import等语句替换成最终输出的可执行的JS语句

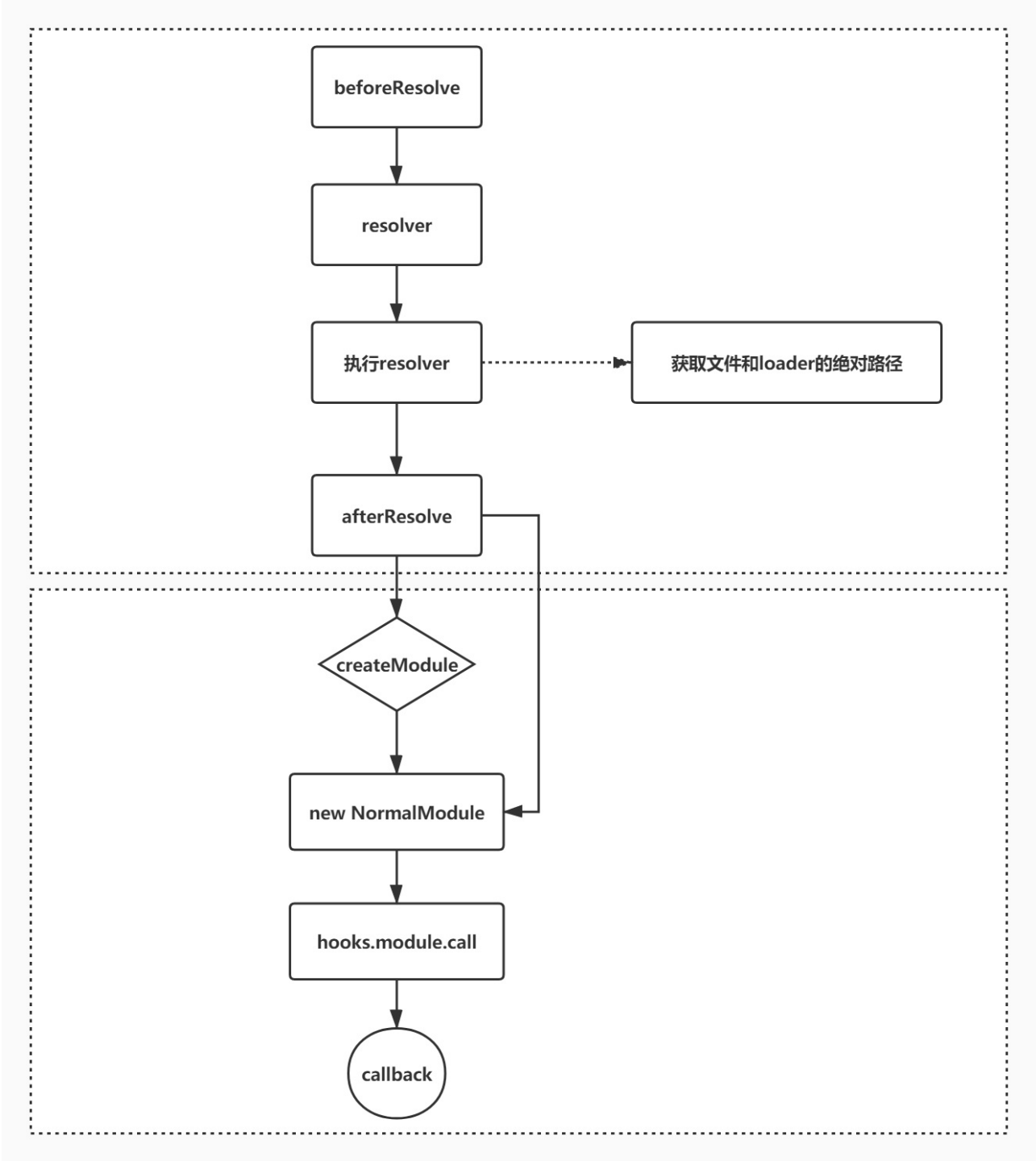
```
module: {
  dependencies: [
    dependency: {
      module
    }
  ]
}
```

7.文件转 module

- create 创建 module 实例
- add module保存到 Compilation 实例上
- build 分析文件内容,并分析依赖项
- processDep 处理依赖，并添加到编译链条中

7.1 create

- NormalModuleFactory.create (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/NormalModuleFactory.js#L373-L414)



7.2 addModule

- add 阶段是将 module 的所有信息保存到 Compilation 中，以便于在最后打包成 chunk 的时候使用
    - 保存到全局的 `Compilation.modules` 数组中
    - 保存到 `Compilation._modules` 对象
    - 添加 `reason`
    - 添加 `Compilation.entries`
- moduleFactory.create.callback (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/Compilation.js#L1073-L1132)

```
addModule(module) {
  const identifier = module.identifier();
  this._modules.set(identifier, module);
  this.modules.push(module);
  this.entries.push(module);
  module.reasons.push(new ModuleReason(module, dependency, explanation));
}
```

7.3 buildModule

- buildModule (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/NormalModule.js#L427-L506)

```
build(options, compilation, resolver, fs, callback) {
  return this.doBuild(options, compilation, resolver, fs, err => {

        const result = this.parser.parse(this._ast || this._source.source());
        handleParseResult(result);
  })
}

doBuild(options, compilation, resolver, fs, callback) {
    runLoaders(
        {
            resource: this.resource,
            loaders: this.loaders,
            context: loaderContext,
            readResource: fs.readFile.bind(fs)
        },
        (err, result) => {

            this._source = this.createSource(
                this.binary ? asBuffer(source) : asString(source),
                resourceBuffer,
                sourceMap
            );
              return callback();
        }
    );
}
```

7.4 parse

- astexplorer (https://astexplorer.net/)
- 将 source 转为 AST(如果 source 是字符串类型)
- 遍历 AST，遇到 import 语句就增加相关依赖
    - 树的遍历 program事件 -> detectStrictMode -> preWalkStatements -> walkStatements
    - 遍历过程中会给 `module`增加 `dependency`实例,每个 `dependency` 类都会有一个 `template` 方法，并且保存了原来代码中的字符位置 `range`,在最后生成打包后的文件时，会用 `template` 的结果替换 `range` 部分的内容
    - 最终得到的 `dependency` 不仅包含了文件中所有的依赖信息，还被用于最终生成打包代码时对原始内容的修改和替换
- parse (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/Parser.js#L2265-L2303)
- HarmonyImportDependencyParserPlugin (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies/HarmonyImportDependencyParserPlugin.js)
- 得到的依赖
    - HarmonyCompatibilityDependency (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies/HarmonyCompatibilityDependency.js) 添加 `__webpack_require__.r(__webpack_exports__)` RuntimeTemplate (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/RuntimeTemplate.js#L333)
    - HarmonyInitDependency (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies/HarmonyInitDependency.js) `var _title_js__WEBPACK_IMPORTED_MODULE_0__`
    - ConstDependency (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies/ConstDependency.js) 放置一个占位符,在最后生成打包文件的时候将其再转为 `use strict`
    - HarmonyImportSideEffectDependency (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies/HarmonyImportSideEffectDependency.js) `var _title_js__WEBPACK_IMPORTED_MODULE_0__ = __webpack_require__(/*! ./title.js */ "./src/title.js");`
    - HarmonyImportSpecifierDependency (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/dependencies/HarmonyImportSpecifierDependency.js) `console.log(_title_js__WEBPACK_IMPORTED_MODULE_0__["message"]);`

title.js

```
export const message = 'zhufeng';
```

lazy.js

```
export const message = 'zhufeng';
```

index.js

```
import { message } from './title.js';
console.log(message);
import('./lazy.js').then(result => {
    console.log(result);
})
console.log(__resourceQuery);
```

```
prewalkStatements(statements) {
    for (let index = 0, len = statements.length; index < len; index++) {
        const statement = statements[index];
        this.prewalkStatement(statement);
    }
}
prewalkImportDeclaration(statement) {
        const source = statement.source.value;
        this.hooks.import.call(statement, source);
        for (const specifier of statement.specifiers) {
            const name = specifier.local.name;
            this.scope.renames.set(name, null);
            this.scope.definitions.add(name);
            switch (specifier.type) {
                case "ImportDefaultSpecifier":
                    this.hooks.importSpecifier.call(statement, source, "default", name);
                    break;
                case "ImportSpecifier":
                    this.hooks.importSpecifier.call(
                        statement,
                        source,
                        specifier.imported.name,
                        name
                    );
                    break;
            }
        }
}
```

HarmonyImportDependencyParserPlugin

```
const sideEffectDep = new HarmonyImportSideEffectDependency(
    source,
    parser.state.module,
    parser.state.lastHarmonyImportOrder,
    parser.state.harmonyParserScope
);
sideEffectDep.loc = statement.loc;
parser.state.module.addDependency(sideEffectDep);
```

7.5 依赖处理

- 对 dependencies 按照代码在文件中出现的先后顺序排序
- Compilation.js (https://github.com/webpack/webpack/blob/c9d4ff7b054fc581c96ce0e53432d44f9dd8ca72/lib/Compilation.js#L1093-L1102)

```
const afterBuild = () => {
    if (addModuleResult.dependencies) {
        this.processModuleDependencies(module, err => {
            if (err) return callback(err);
            callback(null, module);
        });
    } else {
        return callback(null, module);
    }
};
```

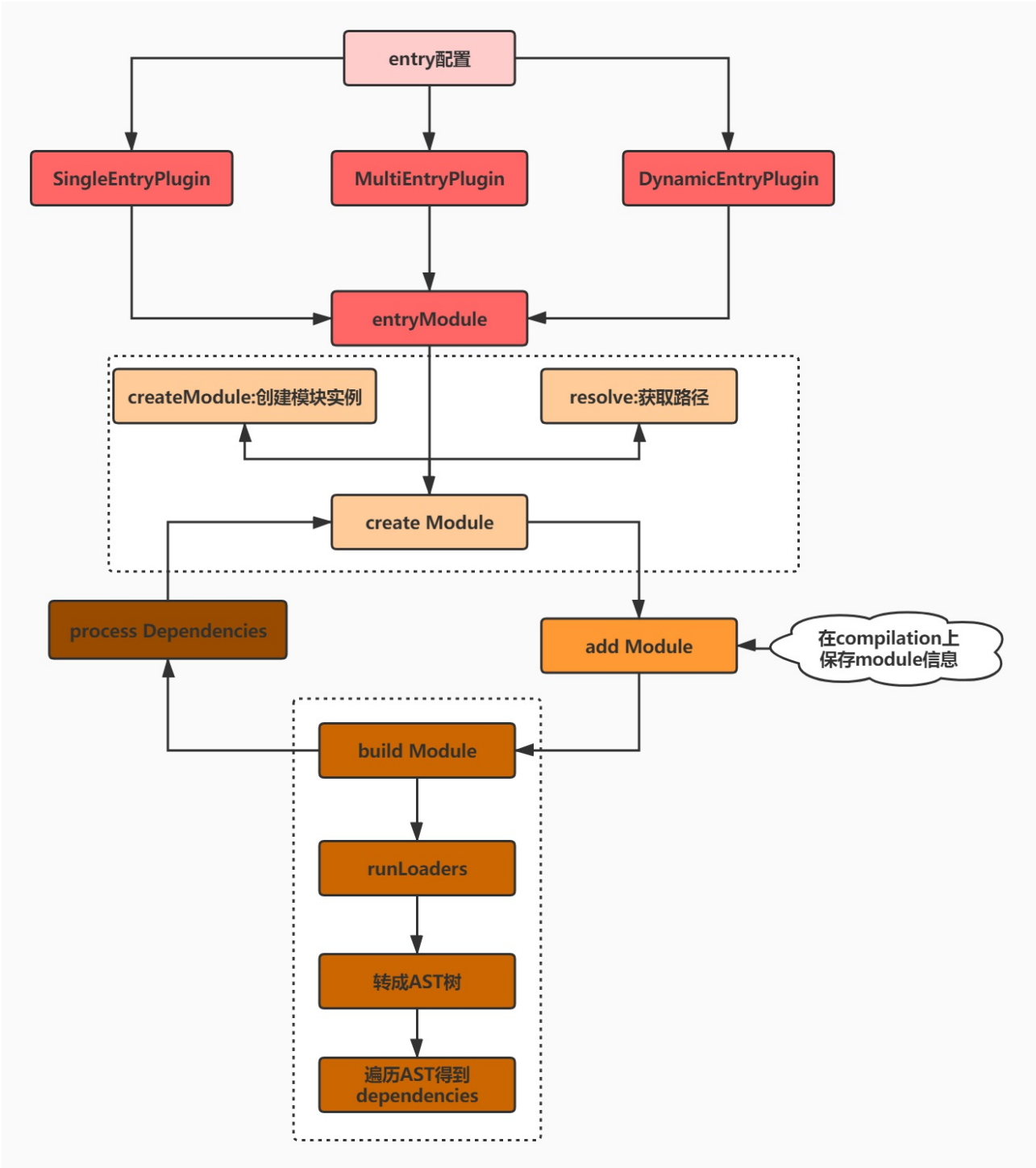
```
dependencies={
    NormalModuleFactory: {
    "module./title.js": [
       HarmonyImportSideEffectDependency,
       HarmonyImportSpecifierDependency
    ],
    "module./lazy.js": [
       HarmonyImportSideEffectDependency,
       HarmonyImportSpecifierDependency
    ]
  }
}
```

```
sortedDependencies = [
  {
    factory: NormalModuleFactory,
    dependencies: [
      HarmonyImportSideEffectDependency,
      HarmonyImportSpecifierDependency
    ]
  },
   {
    factory: NormalModuleFactory,
    dependencies: [
      HarmonyImportSideEffectDependency,
      HarmonyImportSpecifierDependency
    ]
  }
]
```

```
addModuleDependencies(
  module,
  dependencies,
  bail,
  cacheGroup,
  recursive,
  callback
) {
  asyncLib.forEach(
    dependencies,
    (item, callback) => {

      const dependencies = item.dependencies;

       const factory = item.factory;

      factory.create(
          (err, dependentModule) => {
            const addModuleResult = this.addModule(dependentModule);
            dependentModule = addModuleResult.module;

            iterationDependencies(dependencies);

            const afterBuild = () => {

              this.processModuleDependencies(dependentModule, callback);
            };
            this.buildModule(afterBuild);
          }
        );
    }
  );
}
```

```
dependencies{
    HarmonyImportSideEffectDependency(request:"./title.js")
}
blocks{
    ImportDependenciesBlock(request:"./lazy.js")
}
```

7.6 流程


entry配置
SingleEntryPlugin
MultiEntryPlugin
DynamicEntryPlugin
entryModule
createModule:创建模块实例
resolve:获取路径
create Module
process Dependencies
add Module
在compilation上
保存module信息
build Module
runLoaders
转成AST树
遍历AST得到
dependencies