link: null
title: 珠峰架构师成长计划
description: src\index.js
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=152 sentences=1575, words=9677

1.跑通webpack

```
const path = require('path');
module.exports = {
    context: process.cwd(),
    mode: 'development',
    devtool: 'none',
    entry: './src/index.js',
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: '[name].js'
    }
}
```

src\index.js

```
let title = require('./title');
console.log(title);
```

src\title.js

```
module.exports = 'title';
```

```
node cli.js
```

```
const webpack = require("webpack");
const webpackOptions = require("./webpack.config");
const compiler = webpack(webpackOptions);
compiler.run((err, stats) => {
    console.log(err);
    console.log(
        stats.toJson({
            entries: true,
            chunks: true,
            modules: true,
            _modules: true,
            assets: true
        })
    );
});
```

```
{
  errors: [],
  warnings: [],
  version: '4.43.0',
  hash: 'b8d9a2a39e55e9ed6360',
  time: 64,
  builtAt: 1589509767224,
  publicPath: '',
  outputPath: 'C:\\vipdata\\prepare12\\zhufengwebpackprepare\\dist',
  assetsByChunkName: { main: 'main.js' },
  assets: [
    {
      name: 'main.js',
      size: 4126,
      chunks: [Array],
      chunkNames: [Array]
    }
  ],
  entrypoints: {
    main: {
      chunks: [Array],
      assets: [Array],
    }
  },
  namedChunkGroups: {
    main: {
      chunks: [Array],
      assets: [Array]
    }
  },
  chunks: [
    {
      id: 'main',
      rendered: true,
      initial: true,
      entry: true,
      size: 77,
      names: [Array],
      files: [Array],
      hash: '1e1215aa688e72e663af',
      siblings: [],
      parents: [],
      children: [],
      childrenByOrder: [Object: null prototype] {},
      modules: [Array],
      filteredModules: 0,
      origins: [Array]
    }
  ],
  modules: [
    {
      id: './src/index.js',
      identifier: 'C:\\vipdata\\prepare12\\zhufengwebpackprepare\\src\\index.js',
      name: './src/index.js',
      index: 0,
      index2: 1,
      size: 52,
      cacheable: true,
      built: true,
      optional: false,
      prefetched: false,
      chunks: [Array]
      assets: [],
      reasons: [Array],
      source: "let title = require('./title');\r\nconsole.log(title);"
    },
    {
      id: './src/title.js',
      identifier: 'C:\\vipdata\\prepare12\\zhufengwebpackprepare\\src\\title.js',
      name: './src/title.js',
      index: 1,
      index2: 0,
      size: 25,
      cacheable: true,
      built: true,
      optional: false,
      prefetched: false,
      chunks: [Array],
      issuer: 'C:\\vipdata\\prepare12\\zhufengwebpackprepare\\src\\index.js',
      issuerId: './src/index.js',
      issuerName: './src/index.js',
      errors: 0,
      warnings: 0,
      assets: [],
      reasons: [Array],
      source: "module.exports = 'title';"
    }
  ]
}
```


- ^\s*(?=\r?$)\n

```
(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;
  }
  __webpack_require__.m = modules;
  __webpack_require__.c = installedModules;
  __webpack_require__.d = function (exports, name, getter) {
    if (!__webpack_require__.o(exports, name)) {
      Object.defineProperty(exports, name, { enumerable: true, get: getter });
    }
  };
  __webpack_require__.r = function (exports) {
    if (typeof Symbol !== 'undefined' && Symbol.toStringTag) {
      Object.defineProperty(exports, Symbol.toStringTag, { value: 'Module' });
    }
    Object.defineProperty(exports, '__esModule', { value: true });
  };
  __webpack_require__.t = function (value, mode) {
    if (mode & 1) value = __webpack_require__(value);
    if (mode & 8) return value;
    if ((mode & 4) && typeof value === 'object' && value && value.__esModule) return value;
    var ns = Object.create(null);
    __webpack_require__.r(ns);
    Object.defineProperty(ns, 'default', { enumerable: true, value: value });
    if (mode & 2 && typeof value != 'string') for (var key in value) __webpack_require__.d(ns, key, function (key) { return value[key]; }.bind(null, key));
    return ns;
  };
  __webpack_require__.n = function (module) {
    var getter = module && module.__esModule ?
      function getDefault() { return module['default']; } :
      function getModuleExports() { return module; };
    __webpack_require__.d(getter, 'a', getter);
    return getter;
  };
  __webpack_require__.o = function (object, property) { return Object.prototype.hasOwnProperty.call(object, property); };
  __webpack_require__.p = "";
  return __webpack_require__(__webpack_require__.s = "./src/index.js");
})
  ({
    "./src/index.js":
      (function (module, exports, __webpack_require__) {
        let title = __webpack_require__( "./src/title.js");
        console.log(title);
      }),
    "./src/title.js":
      (function (module, exports) {
        module.exports = 'title';
      })
  });
```

2. Compiler.run

```
+const webpack = require("./webpack");
const webpackOptions = require("./webpack.config");
const compiler = webpack(webpackOptions);
compiler.run((err, stats) => {
    console.log(
        stats.toJson({
            entries: true,
            chunks: true,
            modules: true,
            _modules: true,
            assets: true
        })
    );
});
```

webpack\index.js

```
const NodeEnvironmentPlugin = require("./plugins/NodeEnvironmentPlugin");
const Compiler = require("./Compiler");
function webpack(options) {
    options.context = options.context || path.resolve(process.cwd());

    let compiler = new Compiler(options.context);

    compiler.options = Object.assign(compiler.options, options);

    new NodeEnvironmentPlugin().apply(compiler);

    if (options.plugins && Array.isArray(options.plugins)) {
        for (const plugin of options.plugins) {
            plugin.apply(compiler);
        }
    }
    return compiler;
}

module.exports = webpack;
```

webpack\Compiler.js

```
const { Tapable } = require("tapable");
class Compiler extends Tapable {
    constructor(context) {
        super();
        this.options = {};
        this.context = context;
        this.hooks = {};
    }
    run(callback) {
        console.log("Compiler run");
        callback(null, {
            toJson() {
                return {
                    entries: true,
                    chunks: true,
                    modules: true,
                    _modules: true,
                    assets: true
                }
            }
        });
    }
}
module.exports = Compiler;
```

webpack\plugins\NodeEnvironmentPlugin.js

```
const fs = require("fs");
class NodeEnvironmentPlugin {
    apply(compiler) {
        compiler.inputFileSystem = fs;
        compiler.outputFileSystem = fs;
    }
}
module.exports = NodeEnvironmentPlugin;
```

3. 监听make事件

webpack\Compiler.js

```
+const { Tapable, SyncBailHook, AsyncParallelHook } = require("tapable");
class Compiler extends Tapable {
    constructor(context) {
        super();
        this.options = {};
        this.context = context; //设置上下文路径
+       this.hooks = {
+            entryOption: new SyncBailHook(["context", "entry"]),
+            make: new AsyncParallelHook(["compilation"])
+       };
    }
    run(callback) {
        console.log("Compiler run");
        callback(null, {
            toJson() {
                return {
                    entries: true,
                    chunks: true,
                    modules: true,
                    _modules: true,
                    assets: true
                }
            }
        });
    }
}
module.exports = Compiler;
```

webpack\index.js

```
const NodeEnvironmentPlugin = require("./plugins/NodeEnvironmentPlugin");
+const WebpackOptionsApply = require("./WebpackOptionsApply");
const Compiler = require("./Compiler");
function webpack(options) {
    options.context = options.context || path.resolve(process.cwd());
    //创建compiler
    let compiler = new Compiler(options.context);
    //给compiler指定options
    compiler.options = Object.assign(compiler.options, options);
    //插件设置读写文件的API
    new NodeEnvironmentPlugin().apply(compiler);
    //调用配置文件里配置的插件并依次调用
    if (options.plugins && Array.isArray(options.plugins)) {
        for (const plugin of options.plugins) {
            plugin.apply(compiler);
        }
    }
+    new WebpackOptionsApply().process(options, compiler); //处理参数
    return compiler;
}

module.exports = webpack;
```

webpack\WebpackOptionsApply.js

```
const EntryOptionPlugin = require("./plugins/EntryOptionPlugin");
module.exports = class WebpackOptionsApply {
    process(options, compiler) {

        new EntryOptionPlugin().apply(compiler);

        compiler.hooks.entryOption.call(options.context, options.entry);
    }
};
```

webpack\plugins\EntryOptionPlugin.js

```
const SingleEntryPlugin = require("./SingleEntryPlugin");
class EntryOptionPlugin {
    apply(compiler) {
        compiler.hooks.entryOption.tap("EntryOptionPlugin", (context, entry) => {
            new SingleEntryPlugin(context, entry, "main").apply(compiler);
        });
    }
}

module.exports = EntryOptionPlugin;
```

webpack\plugins\SingleEntryPlugin.js

```
class EntryOptionPlugin {
    constructor(context, entry, name) {
        this.context = context;
        this.entry = entry;
        this.name = name;
    }
    apply(compiler) {
        compiler.hooks.make.tapAsync(
            "SingleEntryPlugin",
            (compilation, callback) => {

                const { entry, name, context } = this;
                compilation.addEntry(context, entry, name, callback);
            }
        );
    }
};
module.exports = EntryOptionPlugin;
```

4. make编译

webpack\Compiler.js

```
+const { Tapable, SyncHook, SyncBailHook, AsyncParallelHook, AsyncSeriesHook } = require("tapable");
+const Compilation = require('./Compilation');
+const NormalModuleFactory = require('./NormalModuleFactory');
+const Stats = require('./Stats');
class Compiler extends Tapable {
    constructor(context) {
        super();
        this.options = {};
        this.context = context; //设置上下文路径
        this.hooks = {
            entryOption: new SyncBailHook(["context", "entry"]),
+           make: new AsyncParallelHook(["compilation"]),
+           beforeRun: new AsyncSeriesHook(["compiler"]),
+           run: new AsyncSeriesHook(["compiler"]),
+           beforeCompile: new AsyncSeriesHook(["params"]),
+           compile: new SyncHook(["params"]),
+           make: new AsyncParallelHook(["compilation"]),
+           thisCompilation: new SyncHook(["compilation", "params"]),
+           compilation: new SyncHook(["compilation", "params"]),
+           done: new AsyncSeriesHook(["stats"])
        };
    }
+   run(finalCallback) {
+       //编译完成后的回调
+       const onCompiled = (err, compilation) => {
+           console.log('onCompiled');
+           finalCallback(err, new Stats(compilation));
+       };
+       //准备运行编译
+       this.hooks.beforeRun.callAsync(this, err => {
+           //运行
+           this.hooks.run.callAsync(this, err => {
+               this.compile(onCompiled); //开始编译,编译完成后执行conCompiled回调
+           });
+       });
+   }
+   compile(onCompiled) {
+       const params = this.newCompilationParams();
+       this.hooks.beforeCompile.callAsync(params, err => {
+           this.hooks.compile.call(params);
+           const compilation = this.newCompilation(params);
+           this.hooks.make.callAsync(compilation, err => {
+               console.log('make完成');
+               onCompiled(err, compilation);
+           });
+       });
+   }
+   newCompilationParams() {
+       const params = {
+           normalModuleFactory: new NormalModuleFactory()
+       };
+       return params;
+   }
+   newCompilation(params) {
+       const compilation = new Compilation(this);
+       this.hooks.thisCompilation.call(compilation, params);
+       this.hooks.compilation.call(compilation, params);
+       return compilation;
+   }
}
module.exports = Compiler;
```

webpack\Compilation.js

```
const NormalModuleFactory = require('./NormalModuleFactory');
const { Tapable, SyncHook } = require("tapable");
const Parser = require('./Parser');
const parser = new Parser();
const path = require('path');
class Compilation extends Tapable {
    constructor(compiler) {
        super();
        this.compiler = compiler;
        this.options = compiler.options;
        this.context = compiler.context;
        this.inputFileSystem = compiler.inputFileSystem;
        this.outputFileSystem = compiler.outputFileSystem;
        this.entries = [];
        this.modules = [];
        this.hooks = {
            succeedModule: new SyncHook(["module"])
        }
    }

    addEntry(context, entry, name, callback) {
        this._addModuleChain(context, entry, name, (err, module) => {
            callback(err, module);
        });
    }
    _addModuleChain(context, entry, name, callback) {
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(
            {
                name,
                context: this.context,
                rawRequest: entry,
                resource: path.posix.join(context, entry),
                parser
            });

        this.modules.push(module);
        this.entries.push(module);
        const afterBuild = () => {
            if (module.dependencies) {
                this.processModuleDependencies(module, err => {
                    callback(null, module);
                });
            } else {
                return callback(null, module);
            }
        };
        this.buildModule(module, afterBuild);

    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

webpack\NormalModuleFactory.js

```
const NormalModule = require('./NormalModule');
class NormalModuleFactory {
    create(data) {
        return new NormalModule(data);
    }
}
module.exports = NormalModuleFactory;
```

webpack\NormalModule.js

```
class NormalModule {
    constructor({ name, context, rawRequest, resource, parser }) {
        this.name = name;
        this.context = context;
        this.rawRequest = rawRequest;
        this.resource = resource;
        this.parser = parser;
        this._source = null;
        this._ast = null;
    }

    build(compilation, callback) {
        this.doBuild(compilation, err => {
            this._ast = this.parser.parse(this._source);
            callback();
        });
    }

    doBuild(compilation, callback) {
        let originalSource = this.getSource(this.resource, compilation);
        this._source = originalSource;
        callback();
    }
    getSource(resource, compilation) {
        let originalSource = compilation.inputFileSystem.readFileSync(resource, 'utf8');
        return originalSource;
    }
}
module.exports = NormalModule;
```

webpack\Parser.js

```
const babylon = require('babylon');
const { Tapable } = require("tapable");
class Parser extends Tapable {
    constructor() {
        super();
    }
    parse(source) {
        return babylon.parse(source, { sourceType: 'module', plugins: ['dynamicImport'] });
    }
}
module.exports = Parser;
```

webpack\Stats.js

```
class Stats {
    constructor(compilation) {
        this.entries = compilation.entries;
        this.modules = compilation.modules;
    }
    toJson() {
        return this;
    }
}
module.exports = Stats;
```

5. 编译模块和依赖

webpack\Compilation.js

```
const NormalModuleFactory = require('./NormalModuleFactory');
+const async = require('neo-async');
const { Tapable, SyncHook } = require("tapable");
const Parser = require('./Parser');
const parser = new Parser();
const path = require('path');
class Compilation extends Tapable {
    constructor(compiler) {
        super();
        this.compiler = compiler;
        this.options = compiler.options;
        this.context = compiler.context;
        this.inputFileSystem = compiler.inputFileSystem;
        this.outputFileSystem = compiler.outputFileSystem;
        this.entries = [];
        this.modules = [];
+        this._modules = {};
        this.hooks = {
            succeedModule: new SyncHook(["module"])
        }
    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, callback) {
        this._addModuleChain(context, entry, name, (err, module) => {
            callback(err, module);
        });
    }
    _addModuleChain(context, entry, name, callback) {
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(
            {
                name,  //模块所属的代码块的名称
                context: this.context,//上下文
                rawRequest: entry,
                resource: path.posix.join(context, entry),
                parser
            });//模块完整路径
+        module.moduleId = '.' + path.posix.sep + path.posix.relative(this.context, module.resource);
        this.modules.push(module);
        this.entries.push(module);//把编译好的模块添加到入口列表里面
        const afterBuild = () => {
            if (module.dependencies) {
                this.processModuleDependencies(module, err => {
                    callback(null, module);
                });
            } else {
                return callback(null, module);
            }
        };
        this.buildModule(module, afterBuild);

    }
+    processModuleDependencies(module, callback) {
+        let dependencies = module.dependencies;
+        async.forEach(dependencies, (dependency, done) => {
+            let { name, context, rawRequest, resource, moduleId } = dependency;
+            const moduleFactory = new NormalModuleFactory();
+            let module = moduleFactory.create(
+                {
+                    name,
+                    context,
+                    rawRequest,
+                    moduleId,
+                    resource,
+                    parser
+                });
+            this.modules.push(module);
+            this._modules[module.moduleId] = module;
+            const afterBuild = () => {
+                if (module.dependencies) {
+                    this.processModuleDependencies(module, err => {
+                        done(null, module);
+                    });
+                } else {
+                    return done(null, module);
+                }
+            };
+            this.buildModule(module, afterBuild);
+        }, callback);
+    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

webpack\NormalModule.js

```
+const path = require('path');
+const types = require('babel-types');
+const generate = require('babel-generator').default;
+const traverse = require('babel-traverse').default;
class NormalModule {
+    constructor({ name, context, rawRequest, resource, parser, moduleId }) {
        this.name = name;
        this.context = context;
        this.rawRequest = rawRequest;
        this.resource = resource;
+        this.moduleId = moduleId;
        this.parser = parser;
        this._source = null;
        this._ast = null;
+        this.dependencies = [];
    }
    //解析依赖
    build(compilation, callback) {
        this.doBuild(compilation, err => {
+            let originalSource = this.getSource(this.resource, compilation);
+            // 将 当前模块 的内容转换成 AST
+            const ast = this.parser.parse(originalSource);
+            traverse(ast, {
+                // 如果当前节点是一个函数调用时
+                CallExpression: (nodePath) => {
+                    let node = nodePath.node;
+                    // 当前节点是 require 时
+                    if (node.callee.name === 'require') {
+                        //修改require为__webpack_require__
+                        node.callee.name = '__webpack_require__';
+                        //获取要加载的模块ID
+                        let moduleName = node.arguments[0].value;
+                        //获取扩展名
+                        let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
+                        //获取依赖模块的绝对路径
+                        let dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
+                        //获取依赖模块的模块ID
+                        let dependencyModuleId = '.' + path.posix.sep + path.posix.relative(this.context, dependencyResource);
+                        //添加依赖
+                        this.dependencies.push({
+                            name: this.name, context: this.context, rawRequest: moduleName,
+                            moduleId: dependencyModuleId, resource: dependencyResource
+                        });
+                        node.arguments = [types.stringLiteral(dependencyModuleId)];
+                    }
+                }
+            });
+            let { code } = generate(ast);
+            this._source = code;
+            this._ast = ast;
            callback();
        });
    }
    //获取模块代码
    doBuild(compilation, callback) {
        let originalSource = this.getSource(this.resource, compilation);
        this._source = originalSource;
        callback();
    }
    getSource(resource, compilation) {
        let originalSource = compilation.inputFileSystem.readFileSync(resource, 'utf8');
        return originalSource;
    }
}
module.exports = NormalModule;
```

6. seal

webpack\Compiler.js

```
const { Tapable, SyncHook, SyncBailHook, AsyncParallelHook, AsyncSeriesHook } = require("tapable");
const Compilation = require('./Compilation');
const NormalModuleFactory = require('./NormalModuleFactory');
const Stats = require('./Stats');
class Compiler extends Tapable {
    constructor(context) {
        super();
        this.options = {};
        this.context = context; //设置上下文路径
        this.hooks = {
            entryOption: new SyncBailHook(["context", "entry"]),
            make: new AsyncParallelHook(["compilation"]),
            beforeRun: new AsyncSeriesHook(["compiler"]),
            run: new AsyncSeriesHook(["compiler"]),
            beforeCompile: new AsyncSeriesHook(["params"]),
            compile: new SyncHook(["params"]),
            make: new AsyncParallelHook(["compilation"]),
            thisCompilation: new SyncHook(["compilation", "params"]),
            compilation: new SyncHook(["compilation", "params"]),
+           afterCompile: new SyncHook(["params"]),
            done: new AsyncSeriesHook(["stats"])
        };
    }
    run(finalCallback) {
        //编译完成后的回调
        const onCompiled = (err, compilation) => {
            console.log('onCompiled');
            finalCallback(err, new Stats(compilation));
        };
        //准备运行编译
        this.hooks.beforeRun.callAsync(this, err => {
            //运行
            this.hooks.run.callAsync(this, err => {
                this.compile(onCompiled); //开始编译,编译完成后执行conCompiled回调
            });
        });
    }
    compile(onCompiled) {
        const params = this.newCompilationParams();
        this.hooks.beforeCompile.callAsync(params, err => {
            this.hooks.compile.call(params);
            const compilation = this.newCompilation(params);
            this.hooks.make.callAsync(compilation, err => {
+                compilation.seal(err => {
+                    this.hooks.afterCompile.callAsync(compilation, err => {
+                        return onCompiled(null, compilation);
+                    });
+                });
            });
        });
    }
    newCompilationParams() {
        const params = {
            normalModuleFactory: new NormalModuleFactory()
        };
        return params;
    }
    newCompilation(params) {
        const compilation = new Compilation(this);
        this.hooks.thisCompilation.call(compilation, params);
        this.hooks.compilation.call(compilation, params);
        return compilation;
    }

}
module.exports = Compiler;
```

webpack\Compilation.js

```
const NormalModuleFactory = require('./NormalModuleFactory');
const async = require('neo-async');
const { Tapable, SyncHook } = require("tapable");
const Parser = require('./Parser');
const parser = new Parser();
const path = require('path');
+let Chunk = require('./Chunk');
class Compilation extends Tapable {
    constructor(compiler) {
        super();
        this.compiler = compiler;
        this.options = compiler.options;
        this.context = compiler.context;
        this.inputFileSystem = compiler.inputFileSystem;
        this.outputFileSystem = compiler.outputFileSystem;
        this.entries = [];
        this.modules = [];
        this._modules = {};
        this.chunks = [];
        this.hooks = {
            succeedModule: new SyncHook(["module"]),
+            seal: new SyncHook([]),
+            beforeChunks: new SyncHook([]),
+            afterChunks: new SyncHook(["chunks"])
        }
    }
+    seal(callback) {
+        this.hooks.seal.call();
+        this.hooks.beforeChunks.call();//生成代码块之前
+        for (const module of this.entries) {//循环入口模块
+            const chunk = new Chunk(module);//创建代码块
+            this.chunks.push(chunk);//把代码块添加到代码块数组中
+            //把代码块的模块添加到代码块中
+            chunk.modules = this.modules.filter(module => module.name == chunk.name);
+        }
```

```
+        this.hooks.afterChunks.call(this.chunks);//生成代码块之后
+        callback();//封装结束
+    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, callback) {
        this._addModuleChain(context, entry, name, (err, module) => {
            callback(err, module);
        });
    }
    _addModuleChain(context, entry, name, callback) {
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(
            {
                name,  //模块所属的代码块的名称
                context: this.context,//上下文
                rawRequest: entry,
                resource: path.posix.join(context, entry),
                parser
            });//模块完整路径
        module.moduleId = '.' + path.posix.sep + path.posix.relative(this.context, module.resource);
        this.modules.push(module);
        this.entries.push(module);//把编译好的模块添加到入口列表里面
        const afterBuild = () => {
            if (module.dependencies) {
                this.processModuleDependencies(module, err => {
                    callback(null, module);
                });
            } else {
                return callback(null, module);
            }
        };
        this.buildModule(module, afterBuild);

    }
    processModuleDependencies(module, callback) {
        let dependencies = module.dependencies;
        async.forEach(dependencies, (dependency, done) => {
            let { name, context, rawRequest, resource, moduleId } = dependency;
            const moduleFactory = new NormalModuleFactory();
            let module = moduleFactory.create(
                {
                    name,
                    context,
                    rawRequest,
                    moduleId,
                    resource,
                    parser
                });
            this.modules.push(module);
            this._modules[module.moduleId] = module;
            const afterBuild = () => {
                if (module.dependencies) {
                    this.processModuleDependencies(module, err => {
                        done(null, module);
                    });
                } else {
                    return done(null, module);
                }
            };
            this.buildModule(module, afterBuild);
        }, callback);
    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

webpack\Chunk.js

```
class Chunk {
    constructor(module) {
        this.entryModule = module;
        this.name = module.name;
        this.files = [];
        this.modules = [];
    }
}

module.exports = Chunk;
```

webpack\Stats.js

```
class Stats {
    constructor(compilation) {
        this.entries = compilation.entries;
        this.modules = compilation.modules;
+        this.chunks = compilation.chunks;
    }
    toJson() {
        return this;
    }
}
module.exports = Stats;
```

7.emit

webpack\Compiler.js

```
const { Tapable, SyncHook, SyncBailHook, AsyncParallelHook, AsyncSeriesHook } = require("tapable");
const Compilation = require('./Compilation');
const NormalModuleFactory = require('./NormalModuleFactory');
const Stats = require('./Stats');
+const mkdirp = require('mkdirp');
+const path = require('path');
class Compiler extends Tapable {
    constructor(context) {
        super();
        this.options = {};
        this.context = context; //设置上下文路径
        this.hooks = {
            entryOption: new SyncBailHook(["context", "entry"]),
            make: new AsyncParallelHook(["compilation"]),
            beforeRun: new AsyncSeriesHook(["compiler"]),
            run: new AsyncSeriesHook(["compiler"]),
            beforeCompile: new AsyncSeriesHook(["params"]),
            compile: new SyncHook(["params"]),
            make: new AsyncParallelHook(["compilation"]),
            thisCompilation: new SyncHook(["compilation", "params"]),
            compilation: new SyncHook(["compilation", "params"]),
            afterCompile: new SyncHook(["params"]),
+            emit: new AsyncSeriesHook(["compilation"]),
            done: new AsyncSeriesHook(["stats"])
        };
    }
+    emitAssets(compilation, callback) {
+        const emitFiles = err => {
+            let assets = compilation.assets;
+            for (let file in assets) {
+                let source = assets[file];
+                const targetPath = path.posix.join(this.options.output.path, file);
+                let content = source;
+                this.outputFileSystem.writeFileSync(targetPath, content);
+            }
+            callback();
+        }
+        this.hooks.emit.callAsync(compilation, err => {
+            mkdirp(this.options.output.path, emitFiles);
+        });
+    }
    run(finalCallback) {
        //编译完成后的回调
        const onCompiled = (err, compilation) => {
+            this.emitAssets(compilation, err => {
+                const stats = new Stats(compilation);
+                this.hooks.done.callAsync(stats, err => {
+                    return finalCallback(err, new Stats(compilation));
+                });
+            })
        };
        //准备运行编译
        this.hooks.beforeRun.callAsync(this, err => {
            //运行
            this.hooks.run.callAsync(this, err => {
                this.compile(onCompiled); //开始编译,编译完成后执行conCompiled回调
            });
        });
    }
    compile(onCompiled) {
        const params = this.newCompilationParams();
        this.hooks.beforeCompile.callAsync(params, err => {
            this.hooks.compile.call(params);
            const compilation = this.newCompilation(params);
            this.hooks.make.callAsync(compilation, err => {
                compilation.seal(err => {
                    this.hooks.afterCompile.callAsync(compilation, err => {
                        return onCompiled(null, compilation);
                    });
                });
            });
        });
    }
    newCompilationParams() {
        const params = {
            normalModuleFactory: new NormalModuleFactory()
        };
        return params;
    }
    newCompilation(params) {
        const compilation = new Compilation(this);
        this.hooks.thisCompilation.call(compilation, params);
        this.hooks.compilation.call(compilation, params);
        return compilation;
    }

}
module.exports = Compiler;
```

webpack\Compilation.js

```
const NormalModuleFactory = require('./NormalModuleFactory');
const async = require('neo-async');
const { Tapable, SyncHook } = require("tapable");
const Parser = require('./Parser');
const parser = new Parser();
const path = require('path');
+const Chunk = require('./Chunk');
+const ejs = require('ejs');
+const fs = require('fs');
+const mainTemplate = fs.readFileSync(path.join(__dirname,'template', 'main.ejs'), 'utf8');
+const mainRender = ejs.compile(mainTemplate);
class Compilation extends Tapable {
    constructor(compiler) {
```

```
        super();
        this.compiler = compiler;
        this.options = compiler.options;
        this.context = compiler.context;
        this.inputFileSystem = compiler.inputFileSystem;
        this.outputFileSystem = compiler.outputFileSystem;
        this.entries = [];
        this.modules = [];
        this._modules = {};
        this.chunks = [];
+        this.files = [];  //生成的文件
+        this.assets = {}; //资源
        this.hooks = {
            succeedModule: new SyncHook(["module"]),
            seal: new SyncHook([]),
            beforeChunks: new SyncHook([]),
            afterChunks: new SyncHook(["chunks"])
        }
    }
    seal(callback) {
        this.hooks.seal.call();
        this.hooks.beforeChunks.call();//生成代码块之前
        for (const module of this.entries) {//循环入口模块
            const chunk = new Chunk(module);//创建代码块
            this.chunks.push(chunk);//把代码块添加到代码块数组中
            //把代码块的模块添加到代码块中
            chunk.modules = this.modules.filter(module => module.name == chunk.name);
        }
        this.hooks.afterChunks.call(this.chunks);//生成代码块之后
+       this.createChunkAssets();
        callback();//封装结束
    }
+    createChunkAssets() {
+        for (let i = 0; i < this.chunks.length; i++) {
+            const chunk = this.chunks[i];
+            chunk.files = [];
+            const file = chunk.name + '.js';
+            const source = mainRender({ entryId: chunk.entryModule.moduleId, modules: chunk.modules });
+            chunk.files.push(file);
+            this.emitAsset(file, source);
+        }
+    }
+    emitAsset(file, source) {
+        this.assets[file] = source;
+        this.files.push(file);
+    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, callback) {
        this._addModuleChain(context, entry, name, (err, module) => {
            callback(err, module);
        });
    }
    _addModuleChain(context, entry, name, callback) {
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(
            {
                name,  //模块所属的代码块的名称
                context: this.context,//上下文
                rawRequest: entry,
                resource: path.posix.join(context, entry),
                parser
            });//模块完整路径
        module.moduleId = '.' + path.posix.sep + path.posix.relative(this.context, module.resource);
        this.modules.push(module);
        this.entries.push(module);//把编译好的模块添加到入口列表里面
        const afterBuild = () => {
            if (module.dependencies) {
                this.processModuleDependencies(module, err => {
                    callback(null, module);
                });
            } else {
                return callback(null, module);
            }
        };
        this.buildModule(module, afterBuild);

    }
    processModuleDependencies(module, callback) {
        let dependencies = module.dependencies;
        async.forEach(dependencies, (dependency, done) => {
            let { name, context, rawRequest, resource, moduleId } = dependency;
            const moduleFactory = new NormalModuleFactory();
            let module = moduleFactory.create(
                {
                    name,
                    context,
                    rawRequest,
                    moduleId,
                    resource,
                    parser
                });
            this.modules.push(module);
            this._modules[module.moduleId] = module;
            const afterBuild = () => {
                if (module.dependencies) {
                    this.processModuleDependencies(module, err => {
                        done(null, module);
                    });
                } else {
                    return done(null, module);
                }
            };
            this.buildModule(module, afterBuild);
        }, callback);
```

```
    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

webpack\main.ejs

```
(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 = "");
  })
    ({
        for(let id in modules){
              let {moduleId,_source} = modules[id];%>
              "":
              (function (module, exports,__webpack_require__) {

              }),

    });
```

8.动态import

src\index.js

```
import('./title').then(result => {
    console.log(result.default);
});
```

webpack\Chunk.js

```
class Chunk {
    constructor(module) {
        this.entryModule = module;
        this.name = module.name;
        this.files = [];
        this.modules = [];
+       this.async = module.async;
    }
}

module.exports = Chunk;
```

webpack\plugins\SingleEntryPlugin.js

```
class EntryOptionPlugin {
    constructor(context, entry, name) {
        this.context = context;
        this.entry = entry;
        this.name = name;
    }
    apply(compiler) {
        compiler.hooks.make.tapAsync(
            "SingleEntryPlugin",
            (compilation, callback) => {
                //入口文件 代码块的名称 context上下文绝对路径
                const { entry, name, context } = this;
+                compilation.addEntry(context, entry, name, false, callback);
            }
        );
    }
};
module.exports = EntryOptionPlugin;
```

webpack\Compilation.js

```
const NormalModuleFactory = require('./NormalModuleFactory');
const async = require('neo-async');
const { Tapable, SyncHook } = require("tapable");
const Parser = require('./Parser');
const parser = new Parser();
const path = require('path');
const Chunk = require('./Chunk');
const ejs = require('ejs');
const fs = require('fs');
+const mainTemplate = fs.readFileSync(path.join(__dirname, 'template', 'mainTemplate.ejs'), 'utf8');
+const mainRender = ejs.compile(mainTemplate);
+const chunkTemplate = fs.readFileSync(path.join(__dirname, 'template', 'chunkTemplate.ejs'), 'utf8');
+const chunkRender = ejs.compile(chunkTemplate);
class Compilation extends Tapable {
    constructor(compiler) {
        super();
        this.compiler = compiler;
        this.options = compiler.options;
        this.context = compiler.context;
        this.inputFileSystem = compiler.inputFileSystem;
```

```
        this.outputFileSystem = compiler.outputFileSystem;
        this.entries = [];
        this.modules = [];
        this._modules = {};
        this.chunks = [];
        this.files = [];  //生成的文件
        this.assets = {}; //资源
        this.hooks = {
            succeedModule: new SyncHook(["module"]),
            seal: new SyncHook([]),
            beforeChunks: new SyncHook([]),
            afterChunks: new SyncHook(["chunks"])
        }
    }
    seal(callback) {
        this.hooks.seal.call();
        this.hooks.beforeChunks.call();//生成代码块之前
        for (const module of this.entries) {//循环入口模块
            const chunk = new Chunk(module);//创建代码块
            this.chunks.push(chunk);//把代码块添加到代码块数组中
            //把代码块的模块添加到代码块中
            chunk.modules = this.modules.filter(module => module.name == chunk.name);
        }
        this.hooks.afterChunks.call(this.chunks);//生成代码块之后
        this.createChunkAssets();
        callback();//封装结束
    }
    createChunkAssets() {
        for (let i = 0; i < this.chunks.length; i++) {
            const chunk = this.chunks[i];
            chunk.files = [];
            const file = chunk.name + '.js';
+           let source;
+           if (chunk.async) {
+               source = chunkRender({ chunkName: chunk.name, modules: chunk.modules });
+           } else {
+               source = mainRender({ entryId: chunk.entryModule.moduleId, modules: chunk.modules });
+           }
            chunk.files.push(file);
            this.emitAsset(file, source);
        }
    }
    emitAsset(file, source) {
        this.assets[file] = source;
        this.files.push(file);
    }
    //context ./src/index.js main callback(终级回调)
+   addEntry(context, entry, name, async, callback) {
+       this._addModuleChain(context, entry, name, async, (err, module) => {
            callback(err, module);
        });
    }
+   _addModuleChain(context, entry, name, async, callback) {
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(
            {
                name,  //模块所属的代码块的名称
                context: this.context,//上下文
                rawRequest: entry,
+               async,
                resource: path.posix.join(context, entry),
                parser
            });//模块完整路径
        module.moduleId = '.' + path.posix.sep + path.posix.relative(this.context, module.resource);
        this.modules.push(module);
        this.entries.push(module);//把编译好的模块添加到入口列表里面
        const afterBuild = () => {
            if (module.dependencies) {
                this.processModuleDependencies(module, err => {
                    callback(null, module);
                });
            } else {
                return callback(null, module);
            }
        };
        this.buildModule(module, afterBuild);

    }
    processModuleDependencies(module, callback) {
        let dependencies = module.dependencies;
        async.forEach(dependencies, (dependency, done) => {
+           let { name, context, rawRequest, resource, moduleId, async } = dependency;
            const moduleFactory = new NormalModuleFactory();
            let module = moduleFactory.create(
                {
                    name,
                    context,
                    rawRequest,
                    moduleId,
                    resource,
                    parser,
+                   async
                });
            this.modules.push(module);
            this._modules[module.moduleId] = module;
            const afterBuild = () => {
                if (module.dependencies) {
                    this.processModuleDependencies(module, err => {
                        done(null, module);
                    });
                } else {
                    return done(null, module);
                }
            };
```

```
            this.buildModule(module, afterBuild);
        }, callback);
    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

webpack\NormalModule.js

```
const types = require('babel-types');
const generate = require('babel-generator').default;
const traverse = require('babel-traverse').default;
const path = require('path');
const async = require('neo-async');
class NormalModule {
+    constructor({ name, context, rawRequest, resource, parser, moduleId, async }) {
        this.name = name;
        this.context = context;
        this.rawRequest = rawRequest;
        this.resource = resource;
        this.moduleId = moduleId;
        this.parser = parser;
        this._source = null;
        this._ast = null;
+        this.dependencies = [];
+        this.blocks = [];
+        this.async = async;
    }
    //解析依赖
    build(compilation, callback) {
        this.doBuild(compilation, err => {
            let originalSource = this.getSource(this.resource, compilation);
            // 将 当前模块 的内容转换成 AST
            const ast = this.parser.parse(originalSource);
            traverse(ast, {
                // 如果当前节点是一个函数调用时
                CallExpression: (nodePath) => {
                    let node = nodePath.node;
                    // 当前节点是 require 时
                    if (node.callee.name
                        //修改require为__webpack_require__
                        node.callee.name = '__webpack_require__';
                        //获取要加载的模块ID
                        let moduleName = node.arguments[0].value;
                        //获取扩展名
                        let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
                        //获取依赖模块的绝对路径
                        let dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
                        //获取依赖模块的模块ID
                        let dependencyModuleId = '.' + path.posix.sep + path.posix.relative(this.context, dependencyResource);
                        //添加依赖
                        this.dependencies.push({
                            name: this.name, context: this.context, rawRequest: moduleName,
                            moduleId: dependencyModuleId, resource: dependencyResource
                        });
                        node.arguments = [types.stringLiteral(dependencyModuleId)];
+                    } else if (types.isImport(nodePath.node.callee)) {
+                        //获取要加载的模块ID
+                        let moduleName = node.arguments[0].value;
+                        //获取扩展名
+                        let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
+                        //获取依赖模块的绝对路径
+                        let dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
+                        //获取依赖模块的模块ID
+                        let dependencyModuleId = '.' + path.posix.sep + path.posix.relative(this.context, dependencyResource);
+                        //获取代码块的ID
+                        debugger
+                        let dependencyChunkId = dependencyModuleId.slice(2, dependencyModuleId.lastIndexOf('.')).replace(path.posix.sep, '_', 'g');
+                        // chunkId 不需要带 .js 后缀
+                        nodePath.replaceWithSourceString(`
+                            __webpack_require__.e("${dependencyChunkId}").then(__webpack_require__.t.bind(null,"${dependencyModuleId}",7))
+                        `);
+                        this.blocks.push({
+                            context: this.context,
+                            entry: dependencyModuleId,
+                            name: dependencyChunkId,
+                            async: true
+                        });
                    }
                },
            });
            let { code } = generate(ast);
            this._source = code;
            this._ast = ast;
+           async.forEach(this.blocks, ({ context, entry, name, async }, done) => {
+                compilation._addModuleChain(context, entry, name, async, done);
+           }, callback);
        });
    }
    //获取模块代码
    doBuild(compilation, callback) {
        let originalSource = this.getSource(this.resource, compilation);
        this._source = originalSource;
        callback();
    }
    getSource(resource, compilation) {
        let originalSource = compilation.inputFileSystem.readFileSync(resource, 'utf8');
        return originalSource;
    }
}
module.exports = NormalModule;
```

webpack\mainTemplate.ejs

```
(function (modules) {
  function webpackJsonpCallback(data) {
    var chunkIds = data[0];
    var moreModules = data[1];
    var moduleId, chunkId, i = 0, resolves = [];
    for (; i < chunkIds.length; i++) {
      chunkId = chunkIds[i];
      if (Object.prototype.hasOwnProperty.call(installedChunks, chunkId) && installedChunks[chunkId]) {
        resolves.push(installedChunks[chunkId][0]);
```

```
    }
    installedChunks[chunkId] = 0;
  }
  for (moduleId in moreModules) {
    if (Object.prototype.hasOwnProperty.call(moreModules, moduleId)) {
      modules[moduleId] = moreModules[moduleId];
    }
  }
  if (parentJsonpFunction) parentJsonpFunction(data);
  while (resolves.length) {
    resolves.shift()();
  }
};
var installedModules = {};
var installedChunks = {
  "main": 0
};
function jsonpScriptSrc(chunkId) {
  return __webpack_require__.p + "" + ({}[chunkId] || chunkId) + ".js"
}
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;
}
__webpack_require__.e = function requireEnsure(chunkId) {
  var promises = [];
  var installedChunkData = installedChunks[chunkId];
  if (installedChunkData !== 0) {
    if (installedChunkData) {
      promises.push(installedChunkData[2]);
    } else {
      var promise = new Promise(function (resolve, reject) {
        installedChunkData = installedChunks[chunkId] = [resolve, reject];
      });
      promises.push(installedChunkData[2] = promise);
      var script = document.createElement('script');
      var onScriptComplete;
      script.charset = 'utf-8';
      script.timeout = 120;
      if (__webpack_require__.nc) {
        script.setAttribute("nonce", __webpack_require__.nc);
      }
      script.src = jsonpScriptSrc(chunkId);
      var error = new Error();
      onScriptComplete = function (event) {
        script.onerror = script.onload = null;
        clearTimeout(timeout);
        var chunk = installedChunks[chunkId];
        if (chunk !== 0) {
          if (chunk) {
            var errorType = event && (event.type === 'load' ? 'missing' : event.type);
            var realSrc = event && event.target && event.target.src;
            error.message = 'Loading chunk ' + chunkId + ' failed.\n(' + errorType + ': ' + realSrc + ')';
            error.name = 'ChunkLoadError';
            error.type = errorType;
            error.request = realSrc;
            chunk[1](error);
          }
          installedChunks[chunkId] = undefined;
        }
      };
      var timeout = setTimeout(function () {
        onScriptComplete({ type: 'timeout', target: script });
      }, 120000);
      script.onerror = script.onload = onScriptComplete;
      document.head.appendChild(script);
    }
  }
  return Promise.all(promises);
};
__webpack_require__.m = modules;
__webpack_require__.c = installedModules;
__webpack_require__.d = function (exports, name, getter) {
  if (!__webpack_require__.o(exports, name)) {
    Object.defineProperty(exports, name, { enumerable: true, get: getter });
  }
};

__webpack_require__.r = function (exports) {
  if (typeof Symbol !== 'undefined' && Symbol.toStringTag) {
    Object.defineProperty(exports, Symbol.toStringTag, { value: 'Module' });
  }
  Object.defineProperty(exports, '__esModule', { value: true });
};
__webpack_require__.t = function (value, mode) {
  if (mode & 1) value = __webpack_require__(value);
  if (mode & 8) return value;
  if ((mode & 4) && typeof value === 'object' && value && value.__esModule) return value;
  var ns = Object.create(null);
  __webpack_require__.r(ns);
  Object.defineProperty(ns, 'default', { enumerable: true, value: value });
  if (mode & 2 && typeof value != 'string') for (var key in value) __webpack_require__.d(ns, key, function (key) { return value[key]; }.bind(null, key));
  return ns;
};
__webpack_require__.n = function (module) {
```

```
    var getter = module && module.__esModule ?
      function getDefault() { return module['default']; } :
      function getModuleExports() { return module; };
    __webpack_require__.d(getter, 'a', getter);
    return getter;
  };
  __webpack_require__.o = function (object, property) { return Object.prototype.hasOwnProperty.call(object, property); };
  __webpack_require__.p = "";
  __webpack_require__.oe = function (err) { console.error(err); throw err; };
  var jsonpArray = window["webpackJsonp"] = window["webpackJsonp"] || [];
  var oldJsonpFunction = jsonpArray.push.bind(jsonpArray);
  jsonpArray.push = webpackJsonpCallback;
  jsonpArray = jsonpArray.slice();
  for (var i = 0; i < jsonpArray.length; i++) webpackJsonpCallback(jsonpArray[i]);
  var parentJsonpFunction = oldJsonpFunction;
  return __webpack_require__(__webpack_require__.s = "./src/index.js");
})
  ({
    for(let id in modules){
              let {moduleId,_source} = modules[id];%>
              "":
              (function (module, exports,__webpack_require__) {

              }),

  });
```

webpack\chunkTemplate.ejs

```
(window["webpackJsonp"] = window["webpackJsonp"] || []).push([[""],{

for(let i = 0;i < modules.length;i++){ %>
    "":
    (function(module, exports, __webpack_require__) {

    }),

}]);
```

9.加载第三方模块

```
let _ = require('lodash');
console.log(_.join([1, 2, 3]));
```

webpack\NormalModule.js

```
const types = require('babel-types');
const generate = require('babel-generator').default;
const traverse = require('babel-traverse').default;
const path = require('path');
const async = require('neo-async');
class NormalModule {
    constructor({ name, context, rawRequest, resource, parser, moduleId, async }) {
        this.name = name;
        this.context = context;
        this.rawRequest = rawRequest;
        this.resource = resource;
        this.moduleId = moduleId;
        this.parser = parser;
        this._source = null;
        this._ast = null;
        this.dependencies = [];
        this.blocks = [];
        this.async = async;
    }
    //解析依赖
    build(compilation, callback) {
        this.doBuild(compilation, err => {
            let originalSource = this.getSource(this.resource, compilation);
            // 将 当前模块 的内容转换成 AST
            const ast = this.parser.parse(originalSource);
            traverse(ast, {
                // 如果当前节点是一个函数调用时
                CallExpression: (nodePath) => {
                    let node = nodePath.node;
                    debugger
                    // 当前节点是 require 时
                    if (node.callee.name
                        //修改require为__webpack_require__
                        node.callee.name = '__webpack_require__';
                        //获取要加载的模块ID
                        let moduleName = node.arguments[0].value;
+                        let dependencyResource;
+                        if (moduleName.startsWith('.')) {
+                            //获取扩展名
+                            let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
+                            //获取依赖模块的绝对路径
+                            dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
+                        } else {
+                            dependencyResource = require.resolve(path.posix.join(this.context, 'node_modules', moduleName));
+                            dependencyResource = dependencyResource.replace(/\\/g, path.posix.sep);
+                        }
+                        //获取依赖模块的模块ID
+                        let dependencyModuleId = '.' + dependencyResource.slice(this.context.length);
                        //添加依赖
                        this.dependencies.push({
                            name: this.name, context: this.context, rawRequest: moduleName,
                            moduleId: dependencyModuleId, resource: dependencyResource
                        });
                        node.arguments = [types.stringLiteral(dependencyModuleId)];
                    } else if (types.isImport(nodePath.node.callee)) {
                        //获取要加载的模块ID
                        let moduleName = node.arguments[0].value;
                        //获取扩展名
                        let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
                        //获取依赖模块的绝对路径
                        let dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
                        //获取依赖模块的模块ID
                        let dependencyModuleId = '.' + path.posix.sep + path.posix.relative(this.context, dependencyResource);
                        //获取代码块的ID
                        let dependencyChunkId = dependencyModuleId.slice(2, dependencyModuleId.lastIndexOf('.')).replace(path.posix.sep, '_', 'g');
                        // chunkId 不需要带 .js 后缀
                        nodePath.replaceWithSourceString(`
                            __webpack_require__.e("${dependencyChunkId}").then(__webpack_require__.t.bind(null,"${dependencyModuleId}",7))
                        `);
                        this.blocks.push({
                            context: this.context,
                            entry: dependencyModuleId,
                            name: dependencyChunkId,
                            async: true
                        });
                    }
                },
            });
            let { code } = generate(ast);
            this._source = code;
            this._ast = ast;
            async.forEach(this.blocks, ({ context, entry, name, async }, done) => {
                compilation._addModuleChain(context, entry, name, async, done);
            }, callback);
        });
    }
    //获取模块代码
    doBuild(compilation, callback) {
        let originalSource = this.getSource(this.resource, compilation);
        this._source = originalSource;
        callback();
    }
    getSource(resource, compilation) {
        let originalSource = compilation.inputFileSystem.readFileSync(resource, 'utf8');
        return originalSource;
    }
}
module.exports = NormalModule;
```

10.分离commons和vendor

```
const path = require('path');
module.exports = {
    context: process.cwd(),
    mode: 'development',
    devtool: 'none',
+   entry: {
+       entry1: './src/entry1.js',
+       entry2: './src/entry2.js',
+   },
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: '[name].js'
    }
}
```

src\entry1.js

```
let title = require('./title');
let _ = require('lodash');
console.log(_.upperCase(title));
```

src\entry2.js

```
let title = require('./title');
let _ = require('lodash');
console.log(_.upperCase(title));
```

webpack\plugins\EntryOptionPlugin.js

```
const SingleEntryPlugin = require("./SingleEntryPlugin");
class EntryOptionPlugin {
    apply(compiler) {
        compiler.hooks.entryOption.tap("EntryOptionPlugin", (context, entry) => {
+           if (typeof entry == 'string') {
+               new SingleEntryPlugin(context, entry, 'main').apply(compiler);
+           } else {
+               // 处理多入口
+               for (let entryName in entry) {
+                   new SingleEntryPlugin(context, entry[entryName], entryName).apply(compiler);
+               }
+           }
        });
    }
}
module.exports = EntryOptionPlugin;
```

webpack\Compilation.js

```
const NormalModuleFactory = require('./NormalModuleFactory');
const async = require('neo-async');
const { Tapable, SyncHook } = require("tapable");
const Parser = require('./Parser');
const parser = new Parser();
const path = require('path');
const Chunk = require('./Chunk');
const ejs = require('ejs');
const fs = require('fs');
+const mainTemplate = fs.readFileSync(path.join(__dirname, 'template', 'mainDeferTemplate.ejs'), 'utf8');
const mainRender = ejs.compile(mainTemplate);
const chunkTemplate = fs.readFileSync(path.join(__dirname, 'template', 'chunkTemplate.ejs'), 'utf8');
const chunkRender = ejs.compile(chunkTemplate);
class Compilation extends Tapable {
    constructor(compiler) {
        super();
        this.compiler = compiler;
        this.options = compiler.options;
        this.context = compiler.context;
        this.inputFileSystem = compiler.inputFileSystem;
        this.outputFileSystem = compiler.outputFileSystem;
        this.entries = [];
        this.modules = [];
        this._modules = {};
        this.chunks = [];
        this.files = [];  //生成的文件
        this.assets = {}; //资源
+        this.vendors = [];//第三方模块
+        this.commons = [];//不在node_modules,调用次数大于1的模块
+        this.commonsCountMap = {};//map
        this.hooks = {
            succeedModule: new SyncHook(["module"]),
            seal: new SyncHook([]),
            beforeChunks: new SyncHook([]),
            afterChunks: new SyncHook(["chunks"])
        }
    }
    seal(callback) {
        this.hooks.seal.call();
        this.hooks.beforeChunks.call();//生成代码块之前
+        for (const module of this.modules) {//循环入口模块
+            if (/node_modules/.test(module.moduleId)) {
+                module.name = 'vendors';
+                this.vendors.push(module);
+            } else {
+                if (this.commonsCountMap[module.moduleId]) {
+                    this.commonsCountMap[module.moduleId].count++;
+                } else {
+                    this.commonsCountMap[module.moduleId] = { count: 1, module };
+                }
+            }
+        }
+        for (let moduleId in this.commonsCountMap) {
+            const moduleCount = this.commonsCountMap[moduleId];
+            let { module, count } = moduleCount;
+            if (count >= 2) {
+                module.name = 'commons';
```

```
+                this.commons.push(module);
+            }
+        }
+        let excludeModuleIds = [...this.vendors, ...this.commons].map(item => item.moduleId);
+        this.modules = this.modules.filter(item => !excludeModuleIds.includes(item.moduleId));

        for (const module of this.entries) {//循环入口模块
            const chunk = new Chunk(module);//创建代码块
            this.chunks.push(chunk);//把代码块添加到代码块数组中
            //把代码块的模块添加到代码块中
            chunk.modules = this.modules.filter(module => module.name == chunk.name);
        }
+        if (this.vendors.length) {
+            const chunk = new Chunk(this.vendors[0]);
+            chunk.async = true;
+            this.chunks.push(chunk);
+            chunk.modules = this.vendors;
+        }
+        if (this.commons.length) {
+            const chunk = new Chunk(this.commons[0]);
+            chunk.async = true;
+            this.chunks.push(chunk);
+            chunk.modules = this.commons;
+        }
        this.hooks.afterChunks.call(this.chunks);//生成代码块之后
        this.createChunkAssets();
        callback();//封装结束
    }
    createChunkAssets() {
        for (let i = 0; i < this.chunks.length; i++) {
            const chunk = this.chunks[i];
            chunk.files = [];
            const file = chunk.name + '.js';
            let source;
            if (chunk.async) {
                source = chunkRender({ chunkName: chunk.name, modules: chunk.modules });
            } else {
+                let deferredChunks = [];
+                if (this.commons.length) deferredChunks.push('commons');
+                if (this.vendors.length) deferredChunks.push('vendors');
+                source = mainRender({ entryId: chunk.entryModule.moduleId, modules: chunk.modules, deferredChunks });
            }
            chunk.files.push(file);
            this.emitAsset(file, source);
        }

    }
    emitAsset(file, source) {
        this.assets[file] = source;
        this.files.push(file);
    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, async, callback) {
        this._addModuleChain(context, entry, name, async, (err, module) => {
            callback(err, module);
        });
    }
    _addModuleChain(context, entry, name, async, callback) {
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(
            {
                name,  //模块所属的代码块的名称
                context: this.context,//上下文
                rawRequest: entry,
                async,
                resource: path.posix.join(context, entry),
                parser
            });//模块完整路径
        module.moduleId = '.' + path.posix.sep + path.posix.relative(this.context, module.resource);
        this.modules.push(module);
        this.entries.push(module);//把编译好的模块添加到入口列表里面
        const afterBuild = () => {
            if (module.dependencies) {
                this.processModuleDependencies(module, err => {
                    callback(null, module);
                });
            } else {
                return callback(null, module);
            }
        };
        this.buildModule(module, afterBuild);

    }
    processModuleDependencies(module, callback) {
        let dependencies = module.dependencies;
        async.forEach(dependencies, (dependency, done) => {
            let { name, context, rawRequest, resource, moduleId, async } = dependency;
            const moduleFactory = new NormalModuleFactory();
            let module = moduleFactory.create(
                {
                    name,
                    context,
                    rawRequest,
                    moduleId,
                    resource,
                    parser,
                    async
                });
            this.modules.push(module);
            this._modules[module.moduleId] = module;
            const afterBuild = () => {
                if (module.dependencies) {
                    this.processModuleDependencies(module, err => {
                        done(null, module);
```

```
                    });
                } else {
                    return done(null, module);
                }
            };
            this.buildModule(module, afterBuild);
        }, callback);
    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

webpack\template\mainDeferTemplate.ejs

```
(function (modules) {
  function webpackJsonpCallback(data) {
    var chunkIds = data[0];
    var moreModules = data[1];
    var executeModules = data[2];
    var moduleId, chunkId, i = 0, resolves = [];
    for (; i < chunkIds.length; i++) {
      chunkId = chunkIds[i];
      if (Object.prototype.hasOwnProperty.call(installedChunks, chunkId) && installedChunks[chunkId]) {
        resolves.push(installedChunks[chunkId][0]);
      }
      installedChunks[chunkId] = 0;
    }
    for (moduleId in moreModules) {
      if (Object.prototype.hasOwnProperty.call(moreModules, moduleId)) {
        modules[moduleId] = moreModules[moduleId];
      }
    }
    if (parentJsonpFunction) parentJsonpFunction(data);
    while (resolves.length) {
      resolves.shift()();
    }
    deferredModules.push.apply(deferredModules, executeModules || []);
    return checkDeferredModules();
  };
  function checkDeferredModules() {
    debugger
    var result;
    for (var i = 0; i < deferredModules.length; i++) {
      var deferredModule = deferredModules[i];
      var fulfilled = true;
      for (var j = 1; j < deferredModule.length; j++) {
        var depId = deferredModule[j];
        if (installedChunks[depId] !== 0) fulfilled = false;
      }
      if (fulfilled) {
        deferredModules.splice(i--, 1);
        result = __webpack_require__(__webpack_require__.s = deferredModule[0]);
      }
    }
    return result;
  }
  var installedModules = {};
  var installedChunks = {
    "entry1": 0
  };
  var deferredModules = [];
  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;
  }
  __webpack_require__.m = modules;
  __webpack_require__.c = installedModules;
  __webpack_require__.d = function (exports, name, getter) {
    if (!__webpack_require__.o(exports, name)) {
      Object.defineProperty(exports, name, { enumerable: true, get: getter });
    }
  };
  __webpack_require__.r = function (exports) {
    if (typeof Symbol !== 'undefined' && Symbol.toStringTag) {
      Object.defineProperty(exports, Symbol.toStringTag, { value: 'Module' });
    }
    Object.defineProperty(exports, '__esModule', { value: true });
  };
  __webpack_require__.t = function (value, mode) {
    if (mode & 1) value = __webpack_require__(value);
    if (mode & 8) return value;
    if ((mode & 4) && typeof value === 'object' && value && value.__esModule) return value;
    var ns = Object.create(null);
    __webpack_require__.r(ns);
    Object.defineProperty(ns, 'default', { enumerable: true, value: value });
    if (mode & 2 && typeof value != 'string') for (var key in value) __webpack_require__.d(ns, key, function (key) { return value[key]; }.bind(null, key));
    return ns;
  };
  __webpack_require__.n = function (module) {
    var getter = module && module.__esModule ?
```

```
      function getDefault() { return module['default']; } :
      function getModuleExports() { return module; };
    __webpack_require__.d(getter, 'a', getter);
    return getter;
  };
  __webpack_require__.o = function (object, property) { return Object.prototype.hasOwnProperty.call(object, property); };
  __webpack_require__.p = "";
  var jsonpArray = window["webpackJsonp"] = window["webpackJsonp"] || [];
  var oldJsonpFunction = jsonpArray.push.bind(jsonpArray);
  jsonpArray.push = webpackJsonpCallback;
  jsonpArray = jsonpArray.slice();
  for (var i = 0; i < jsonpArray.length; i++) webpackJsonpCallback(jsonpArray[i]);
  var parentJsonpFunction = oldJsonpFunction;
  deferredModules.push([""0?","'+deferredChunks.join('","')+'"':""%>]);
  return checkDeferredModules();
})
  ({
    for(let id in modules){
              let {moduleId,_source} = modules[id];%>
              "":
              (function (module, exports,__webpack_require__) {

              }),

  });
```

11.支持loader

```
const path = require('path');
module.exports = {
    context: process.cwd(),
    mode: 'development',
    devtool: 'none',
+    entry: './src/index.js',
    module: {
        rules: [
            {
                test: /\.less$/,
                use: ['style-loader', 'less-loader']
            }
        ]
    },
    output: {
        path: path.resolve(__dirname, 'dist'),
        filename: '[name].js'
    }
}
```

src\index.js

```
+require('./index.less');
let title = require('./title');
let _ = require('lodash');
console.log(_.upperCase(title));
```

webpack\NormalModule.js

```
const types = require('babel-types');
const generate = require('babel-generator').default;
const traverse = require('babel-traverse').default;
const path = require('path');
const async = require('neo-async');
const runLoaders = require('./loader-runner');
const fs = require('fs');
class NormalModule {
    constructor({ name, context, rawRequest, resource, parser, moduleId, async }) {
        this.name = name;
        this.context = context;
        this.rawRequest = rawRequest;
        this.resource = resource;
        this.moduleId = moduleId;
        this.parser = parser;
        this._source = null;
        this._ast = null;
        this.dependencies = [];
        this.blocks = [];
        this.async = async;
    }
    //解析依赖
    build(compilation, callback) {
        this.doBuild(compilation, err => {
+            const afterSource = (err, source) => {
               // 将 当前模块 的内容转换成 AST
               const ast = this.parser.parse(source);
               traverse(ast, {
                   // 如果当前节点是一个函数调用时
                   CallExpression: (nodePath) => {
                       let node = nodePath.node;
                       // 当前节点是 require 时
                       if (node.callee.name
                           //修改require为__webpack_require__
                           node.callee.name = '__webpack_require__';
                           //获取要加载的模块ID
                           let moduleName = node.arguments[0].value;
                           let dependencyResource;
                           if (moduleName.startsWith('.')) {
                               //获取扩展名
                               let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
                               //获取依赖模块的绝对路径
                               dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
                           } else {
                               dependencyResource = require.resolve(path.posix.join(this.context, 'node_modules', moduleName));
                               dependencyResource = dependencyResource.replace(/\\/g, path.posix.sep);
                           }
```

```
                            //获取依赖模块的模块ID
                            let dependencyModuleId = '.' + dependencyResource.slice(this.context.length);
                            //添加依赖
                            this.dependencies.push({
                                name: this.name, context: this.context, rawRequest: moduleName,
                                moduleId: dependencyModuleId, resource: dependencyResource
                            });
                            node.arguments = [types.stringLiteral(dependencyModuleId)];
                        } else if (types.isImport(nodePath.node.callee)) {
                            //获取要加载的模块ID
                            let moduleName = node.arguments[0].value;
                            //获取扩展名
                            let extension = moduleName.split(path.posix.sep).pop().indexOf('.') == -1 ? '.js' : '';
                            //获取依赖模块的绝对路径
                            let dependencyResource = path.posix.join(path.posix.dirname(this.resource), moduleName + extension);
                            //获取依赖模块的模块ID
                            let dependencyModuleId = '.' + path.posix.sep + path.posix.relative(this.context, dependencyResource);
                            //获取代码块的ID
                            let dependencyChunkId = dependencyModuleId.slice(2, dependencyModuleId.lastIndexOf('.')).replace(path.posix.sep, '_', 'g');
                            // chunkId 不需要带 .js 后缀
                            nodePath.replaceWithSourceString(`
                __webpack_require__.e("${dependencyChunkId}").then(__webpack_require__.t.bind(null,"${dependencyModuleId}",7))
            `);
                            this.blocks.push({
                                context: this.context,
                                entry: dependencyModuleId,
                                name: dependencyChunkId,
                                async: true
                            });
                        }
                    },
                });
                let { code } = generate(ast);
                this._source = code;
                this._ast = ast;
                async.forEach(this.blocks, ({ context, entry, name, async }, done) => {
                    compilation._addModuleChain(context, entry, name, async, done);
                }, callback);
            }
+           this.getSource(this.resource, compilation, afterSource);
        });
    }
    //获取模块代码
    doBuild(compilation, callback) {
+       this.getSource(this.resource, compilation, (err, source) => {
+           this._source = source;
+           callback();
+       });
    }
+    getSource(resource, compilation, callback) {
+       let { module: { rules } } = compilation.options;
+       let loaders = [];
+       for (let i = 0; i < rules.length; i++) {
+           let rule = rules[i];
+           if (rule.test.test(resource)) {
+               let useLoaders = rule.use;
+               loaders = [...loaders, ...useLoaders];
+           }
+       }
+       loaders = loaders.map(loader => require.resolve(path.posix.join(this.context, 'loaders', loader)));
+       let source = runLoaders({
+           resource,
+           loaders,
+           context: {},
+           readResource: fs
+       }, function (err, result) {
+           callback(err, result);
+       });
+       return source;
    }
}
module.exports = NormalModule;
```

loaders\less-loader.js

```
var less = require('less');
module.exports = function (source) {
    let css;
    less.render(source, (err, output) => {
        css = output.css;
    });
    return css;
}
```

loaders\style-loader.js

```
module.exports = function (source) {
    let str = `
      let style = document.createElement('style');
      style.innerHTML = ${JSON.stringify(source)};
      document.head.appendChild(style);
    `;
    return str;
}
```

src\index.less

```
@color:red;
body{
    background-color:@color;
}
```

webpack\loader-runner.js

```
let fs = require('fs');
```

```
let path = require('path');
function createLoaderObject(loader) {
    let obj = {};
    obj.data = {};
    obj.request = loader;
    obj.normal = require(loader);
    obj.pitch = obj.normal.pitch;
    return obj;
}
function runLoaders(options, callback) {
    debugger
    var resource = options.resource || "";
    var loaders = options.loaders || [];
    var loaderContext = options.context || {};
    var readResource = options.readResource || fs;

    loaders = loaders.map(createLoaderObject);
    loaderContext.loaderIndex = 0;
    loaderContext.readResource = readResource;
    loaderContext.resource = resource;
    loaderContext.loaders = loaders;
    let isSync = true;
    let innerCallback = loaderContext.callback = function (err, args) {
        isSync = true;
        loaderContext.loaderIndex--;
        iterateNormalLoaders(loaderContext, args, callback);

    }
    loaderContext.async = function () {
        isSync = false;
        return innerCallback;
    }

    Object.defineProperty(loaderContext, "request", {
        get() {
            return loaderContext.loaders.map(item => item.request)
                .concat(loaderContext.resource).join('!')
        }
    })
    Object.defineProperty(loaderContext, "remainingRequest", {
        get() {
            return loaderContext.loaders
                .slice(loaderContext.loaderIndex + 1)
                .map(item => item.request)
                .concat(loaderContext.resource).join('!')
        }
    })

    Object.defineProperty(loaderContext, "currentRequest", {
        get() {
            return loaderContext.loaders
                .slice(loaderContext.loaderIndex)
                .map(item => item.request)
                .concat(loaderContext.resource).join('!')
        }
    })
    Object.defineProperty(loaderContext, "previousRequest", {
        get() {
            return loaderContext.loaders
                .slice(0, loaderContext.loaderIndex)
                .map(item => item.request).join('!')
        }
    })
    Object.defineProperty(loaderContext, "data", {
        get() {
            return loaderContext.loaders[loaderContext.loaderIndex].data;
        }
    })
    iteratePitchingLoaders(loaderContext, callback);
    function processResource(loaderContext, callback) {
        let buffer = loaderContext.readResource.readFileSync(loaderContext.resource, 'utf8');
        iterateNormalLoaders(loaderContext, buffer, callback);
    }
    function iterateNormalLoaders(loaderContext, args, callback) {
        if (loaderContext.loaderIndex < 0) {
            return callback(null, args);
        }
        let currentLoaderObject = loaderContext.loaders[loaderContext.loaderIndex];
        let fn = currentLoaderObject.normal;
        args = fn.apply(loaderContext, [args]);

        if (isSync) {
            loaderContext.loaderIndex--;
            iterateNormalLoaders(loaderContext, args, callback);
        }
    }
    function iteratePitchingLoaders(loaderContext, callback) {

        if (loaderContext.loaderIndex >= loaderContext.loaders.length) {
            loaderContext.loaderIndex--;
            return processResource(loaderContext, callback);
        }
        let currentLoaderObject = loaderContext.loaders[loaderContext.loaderIndex];
        let fn = currentLoaderObject.pitch;
        if (!fn) {
            loaderContext.loaderIndex++;
            return iteratePitchingLoaders(loaderContext, callback);
        }
        let args = fn.apply(loaderContext, [loaderContext.remindingRequest, loaderContext.previousRequest, loaderContext.data]);
        if (args) {
            loaderContext.loaderIndex--;
            iterateNormalLoaders(loaderContext, args, callback);
        } else {
            loaderContext.loaderIndex++;
```

```
            iteratePitchingLoaders(loaderContext, callback);
        }
    }
}
module.exports = runLoaders;
```