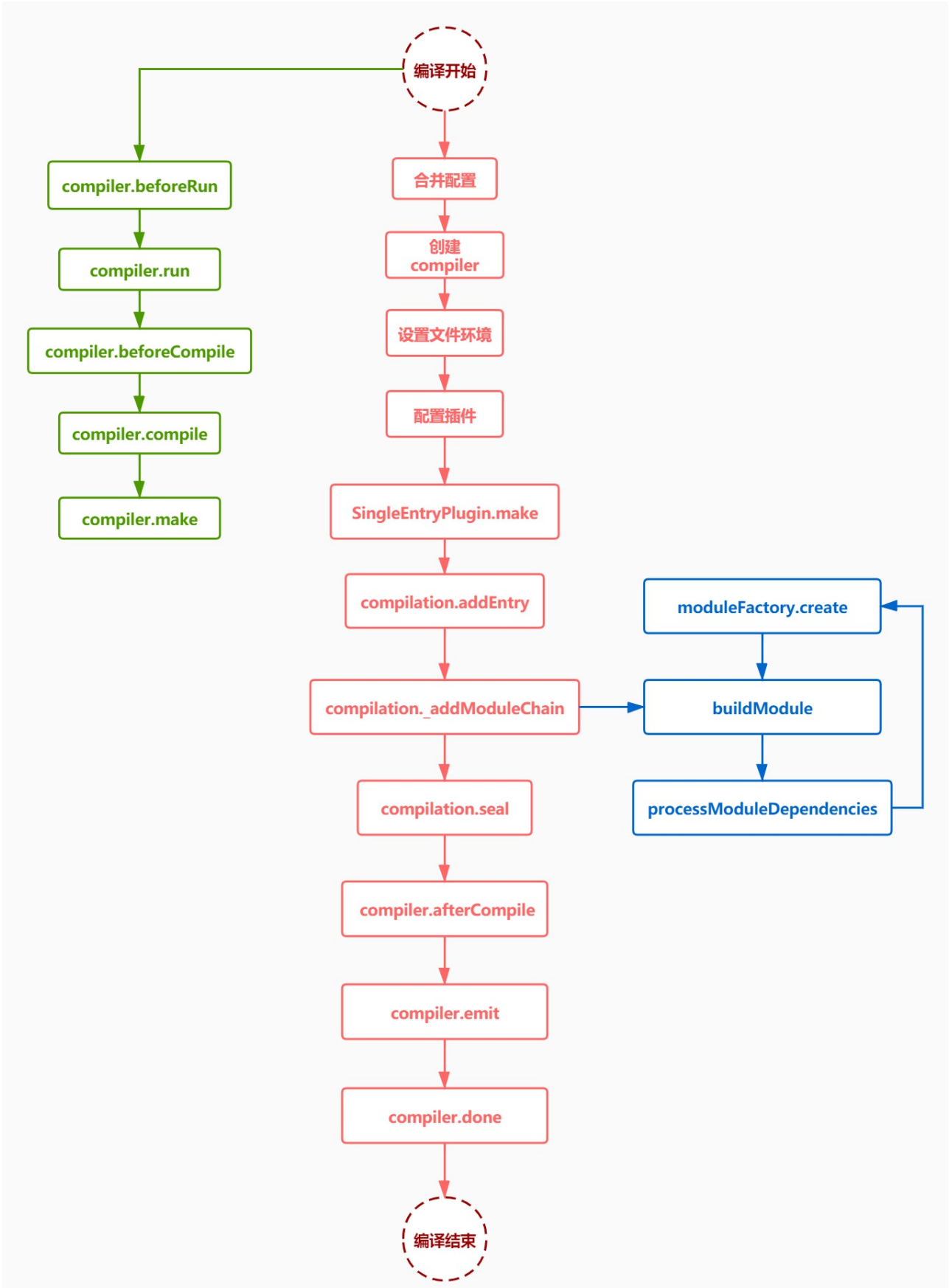
link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=222 sentences=1860, words=10839

1.跑通webpack



1.1 webpack.config.js #

```
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'
    }
}
```

1.2 src\index.js

src\index.js

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

1.3 src\title.js

src\title.js

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

1.4 cli.js

```
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,
            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';"
    }
  ]
}
```


1.5 main.js

- ^\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

2.1 cli.js

```
+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,
            assets: true
        })
    );
});
```

2.2 webpack\index.js

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;
```

2.3 Compiler.js

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,
                    assets: true
                }
            }
        });
    }
}
module.exports = Compiler;
```

2.4 NodeEnvironmentPlugin.js

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

3.1 Compiler.js

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,
                    assets: true
                }
            }
        });
    }
}
module.exports = Compiler;
```

3.2 webpack\index.js

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;
```

3.3 WebpackOptionsApply.js

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);
    }
};
```

3.4 EntryOptionPlugin.js

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;
```

3.5 SingleEntryPlugin.js

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

4.1 Compiler.js

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"]),
+           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;
```

4.2 Compilation.js

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;
```

4.3 NormalModuleFactory.js

webpack\NormalModuleFactory.js

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

4.4 NormalModule.js

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;
```

4.5 Parser.js

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;
```

4.6 Stats.js

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

5.1 webpack\Compilation.js

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.hooks = {
            succeedModule: new SyncHook(["module"])
        }
    }
    //context ./src/index.js main callback(终级回调)
+    _addModuleChain(context,entry,name,callback){
+        this.createModule({
+            name,//所属的代码块的名称 main
+            context:this.context,//上下文
+            rawRequest:entry,// ./src/index.js
+            resource:path.posix.join(context,entry),//此模块entry的的绝对路径
+            parser,
+        },module=>{this.entries.push(module)},callback);
+    }
+    createModule(data,addEntry,callback){
+        //先创建模块工厂
+        const moduleFactory = new NormalModuleFactory();
+        let module = moduleFactory.create(data);
+        //非常非常重要 模块的ID如何生成？模块的ID是一个相对于根目录的相对路径
+        //index.js ./src/index.js title.js ./src/title.js
+        //relative返回一个相对路径 从根目录出出到模块的绝地路径 得到一个相对路径
+        module.moduleId = '.'+path.posix.sep+path.posix.relative(this.context,module.resource);
+        addEntry&&addEntry(module);
+        this.modules.push(module);//把模块添加到完整的模块数组中
+        const afterBuild = (err,module)=>{
+            if(module.dependencies){//如果一个模块编译完成,发现它有依赖的模块,那么递归编译它的依赖模块
+                this.processModuleDependencies(module,(err)=>{
+                    //当这个入口模块和它依赖的模块都编译完成了,才会让调用入口模块的回调
+                    callback(err,module);
+                });
+            }else{
+                callback(err,module);
+            }
+        }
+        this.buildModule(module,afterBuild);
+    }
+    processModuleDependencies(module,callback){
+        let dependencies= module.dependencies;
+        //因为我希望可以并行的同时开始编译依赖的模块,然后等所有依赖的模块全部编译完成后才结束
+        async.forEach(dependencies,(dependency,done)=>{
+            let {name,context,rawRequest,resource,moduleId} = dependency;
+            this.createModule({
+                name,
+                context,
+                rawRequest,
+                resource,
+                moduleId,
+                parser
+            },null,done);
+        },callback);
+    }
    buildModule(module,afterBuild){
        module.build(this,(err)=>{
            this.hooks.succeedModule.call(module)
            afterBuild(null,module);
        });
    }
}
module.exports = Compilation;
```

5.2 NormalModule.js

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||('./'+path.posix.relative(context,resource));
        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

6.1 Compiler.js

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"]),
            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 AsyncSeriesHook(["compilation"]),
            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;
```

6.2 Compilation.js

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.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)
    _addModuleChain(context,entry,name,callback){
        this.createModule({
            name,//所属的代码块的名称 main
            context:this.context,//上下文
            rawRequest:entry,// ./src/index.js
            resource:path.posix.join(context,entry),//此模块entry的的绝对路径
            parser,
        },module=>{this.entries.push(module)},callback);
    }
    createModule(data,addEntry,callback){
        //先创建模块工厂
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(data);
        //非常非常重要 模块的ID如何生成？模块的ID是一个相对于根目录的相对路径
        //index.js ./src/index.js title.js ./src/title.js
        //relative返回一个相对路径 从根目录出出到模块的绝地路径 得到一个相对路径
        module.moduleId = '.'+path.posix.sep+path.posix.relative(this.context,module.resource);
        addEntry&&addEntry(module);
        this.modules.push(module);//把模块添加到完整的模块数组中
        const afterBuild = (err,module)=>{
            if(module.dependencies){//如果一个模块编译完成,发现它有依赖的模块,那么递归编译它的依赖模块
                this.processModuleDependencies(module,(err)=>{
                    //当这个入口模块和它依赖的模块都编译完成了,才会让调用入口模块的回调
                    callback(err,module);
                });
            }else{
                callback(err,module);
            }
        }
        this.buildModule(module,afterBuild);
    }
    processModuleDependencies(module,callback){
        let dependencies= module.dependencies;
        //因为我希望可以并行的同时开始编译依赖的模块,然后等所有依赖的模块全部编译完成后才结束
        async.forEach(dependencies,(dependency,done)=>{
            let {name,context,rawRequest,resource,moduleId} = dependency;
            this.createModule({
                name,
                context,
                rawRequest,
                resource,
                moduleId,
                parser
            },null,done);
        },callback);
    }
    buildModule(module,afterBuild){
        module.build(this,(err)=>{
            this.hooks.succeedModule.call(module)
            afterBuild(null,module);
        });
    }
}
module.exports = Compilation;
```

6.3 webpack\Chunk.js

webpack\Chunk.js

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

module.exports = Chunk;
```

6.4 Stats.js

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

7.1 Compilation.js

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.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, finalCallback) {
        this._addModuleChain(context, entry, name, (err, module) => {
        finalCallback(err, module);
        });
    }
    _addModuleChain(context, rawRequest, name, callback) {
        this.createModule({
        name,context,rawRequest,parser,
        resource:path.posix.join(context,rawRequest),
        moduleId:'./'+path.posix.relative(context,path.posix.join(context,rawRequest))
        },entryModule=>this.entries.push(entryModule),callback);
    }
```

```
    /**
    * 创建并编译一个模块
    * @param {*} data 要编译的模块信息
    * @param {*} addEntry  可选的增加入口的方法 如果这个模块是入口模块,如果不是的话,就什么不做
    * @param {*} callback 编译完成后可以调用callback回调
    */
    createModule(data, addEntry, callback) {
        //通过模块工厂创建一个模块
        let module = normalModuleFactory.create(data);
        addEntry&&addEntry(module);//如果是入口模块,则添加入口里去
        this.modules.push(module);//给普通模块数组添加一个模块
        const afterBuild = (err, module) => {
        //如果大于0,说明有依赖
        if (module.dependencies.length > 0) {
            this.processModuleDependencies(module, err => {
            callback(err, module);
            });
        } else {
            callback(err, module);
        }
        }
        this.buildModule(module, afterBuild);
    }
    /**
    * 处理编译模块依赖
    * @param {*} module ./src/index.js
    * @param {*} callback
    */
    processModuleDependencies(module, callback) {
        //1.获取当前模块的依赖模块
        let dependencies = module.dependencies;
        //遍历依赖模块,全部开始编译,当所有的依赖模块全部编译完成后才调用callback
        async.forEach(dependencies, (dependency, done) => {
        let { name, context, rawRequest, resource, moduleId } = dependency;
        this.createModule({
            name,context,rawRequest,parser,
            resource,moduleId
        },null,done);
        }, callback);
    }
    buildModule(module,afterBuild){
        module.build(this,(err)=>{
            this.hooks.succeedModule.call(module)
            afterBuild(null,module);
        });
    }
}
module.exports = Compilation;
```

7.2 Compiler.js

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"]),
            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 AsyncSeriesHook(["compilation"]),
+            emit: new AsyncSeriesHook(["compilation"]),
            done: new AsyncSeriesHook(["stats"])
        };
    }
+    emitAssets(compilation, callback) {
+        const emitFiles = (err)=>{
+            const assets = compilation.assets;
+            let outputPath = this.options.output.path;//dist
+            for(let file in assets){
+              let source = assets[file];//得到文件名和文件内容
+              let targetPath = path.posix.join(outputPath,file);//得到输出的路径 targetPath
+              this.outputFileSystem.writeFileSync(targetPath,source,'utf8');//NodeEnvironmentPlugin
+            }
+          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=>{
+                let stats = new Stats(compilation);//stats是一 个用来描述打包后结果的对象
+                this.hooks.done.callAsync(stats,err=>{//done表示整个流程结束了
+                callback(err,stats);
+                });
+            });
        };
        //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;
```

7.3 main.ejs

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__("");
  })
    ({
      for(let module of modules)
        {%>
            "":
            (function (module, exports, __webpack_require__) {
              module._source%>
            }),

    });
```

8.动态import

8.1 webpack.config.js

```
    output:{
        path:path.resolve(__dirname,'dist'),
        filename:'[name].js',
+        chunkFilename:'[name].js'
    }
```

8.2 src\index.js

src\index.js

```
require('./sync');

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

8.3 Chunk.js

webpack\Chunk.js

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

module.exports = Chunk;
```

8.4 Compilation.js

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.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 entryModule of this.entries) {//循环入口模块
            const chunk = new Chunk(entryModule);//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({ entryModuleId: 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, finalCallback) {
        this._addModuleChain(context, entry, name,false, (err, module) => {
        finalCallback(err, module);
        });
    }
    _addModuleChain(context, rawRequest, name,async, callback) {
        this.createModule({
        name,context,rawRequest,parser,
        resource:path.posix.join(context,rawRequest),
        moduleId:'./'+path.posix.relative(context,path.posix.join(context,rawRequest)),
        async
        },entryModule=>this.entries.push(entryModule),callback);
    }
    /**
    * 创建并编译一个模块
    * @param {*} data 要编译的模块信息
    * @param {*} addEntry  可选的增加入口的方法 如果这个模块是入口模块,如果不是的话,就什么不做
    * @param {*} callback 编译完成后可以调用callback回调
    */
    createModule(data, addEntry, callback) {
        //通过模块工厂创建一个模块
        let module = normalModuleFactory.create(data);
        addEntry&&addEntry(module);//如果是入口模块,则添加入口里去
        this.modules.push(module);//给普通模块数组添加一个模块
        const afterBuild = (err, module) => {
        //如果大于0,说明有依赖
        if (module.dependencies.length > 0) {
            this.processModuleDependencies(module, err => {
            callback(err, module);
            });
        } else {
            callback(err, module);
        }
        }
        this.buildModule(module, afterBuild);
    }
    /**
    * 处理编译模块依赖
    * @param {*} module ./src/index.js
    * @param {*} callback
    */
    processModuleDependencies(module, callback) {
        //1.获取当前模块的依赖模块
        let dependencies = module.dependencies;
        //遍历依赖模块,全部开始编译,当所有的依赖模块全部编译完成后才调用callback
        async.forEach(dependencies, (dependency, done) => {
        let { name, context, rawRequest, resource, moduleId } = dependency;
        this.createModule({
            name,context,rawRequest,parser,
            resource,moduleId
        },null,done);
        }, callback);
    }
    buildModule(module, afterBuild) {
        module.build(this, (err) => {
            this.hooks.succeedModule.call(module);
            return afterBuild();
        });
    }
}
module.exports = Compilation;
```

8.5 NormalModule.js

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||('./'+path.posix.relative(context,resource));
        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
+                       let chunkName = compilation.asyncChunkCounter++;
+                        if(Array.isArray(node.arguments[0].leadingComments)&&
+                        node.arguments[0].leadingComments.length>0){
+                            let leadingComments = node.arguments[0].leadingComments[0].value;
+                            let regexp = /webpackChunkName:\s*['"]([^'"]+)['"]/;
+                            chunkName = leadingComments.match(regexp)[1];
+                        }
+                        nodePath.replaceWithSourceString(`__webpack_require__.e("${chunkName}").then(__webpack_require__.t.bind(null,"${depModuleId}", 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;
```

8.6 webpack\mainTemplate.ejs

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 + "" + ({ "sum": "sum", "title": "title" }[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 = "");
})
  ({
      for(let module of modules)
      {%>
          "":
          (function (module, exports, __webpack_require__) {
            module._source%>
          }),

  });
```

8.7 chunkTemplate.ejs

webpack\chunkTemplate.ejs

```
(window["webpackJsonp"] = window["webpackJsonp"] || []).push([[""], {
    for(let module of modules)
    {%>
        "":
        (function (module, exports, __webpack_require__) {
          module._source%>
        }),

}]);
```

8.8 dist\main.js

dist\main.js

```
(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 + "" + ({ "sum": "sum", "title": "title" }[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");
})
  ({

    "./src/index.js":
      (function (module, exports, __webpack_require__) {
        __webpack_require__("./src/sync.js");

        __webpack_require__.e("title").then(__webpack_require__.t.bind(null, "./src/title.js", 7)).then(result => {
          console.log(result.default);
        });
        __webpack_require__.e("sum").then(__webpack_require__.t.bind(null, "./src/sum.js", 7)).then(result => {
          console.log(result.default);
        });
      }),

    "./src/sync.js":
      (function (module, exports, __webpack_require__) {
        module.exports = 'sync';
      }),

  });
```

8.9 sum.js

dist\sum.js

```
(window["webpackJsonp"] = window["webpackJsonp"] || []).push([["sum"], {
        "./src/sum.js":
        (function (module, exports, __webpack_require__) {
          module.exports = 'sum';
        }),
  }]);
```

8.10 title.js

dist\title.js

```
(window["webpackJsonp"] = window["webpackJsonp"] || []).push([["title"], {
  "./src/title.js":
    (function (module, exports, __webpack_require__) {
      let inner_title = __webpack_require__("./src/inner_title.js");
      module.exports = inner_title;
    }),
  "./src/inner_title.js":
    (function (module, exports, __webpack_require__) {
      module.exports = 'inner_title';
    }),
}]);
```

9.加载第三方模块

9.1 src\index.js

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

9.2 NormalModule.js

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||('./'+path.posix.relative(context,resource));
        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

10.1 webpack.config.js

```
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'
    }
}
```

10.2 src\entry1.js

src\entry1.js

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

10.3 src\entry2.js

src\entry2.js

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

10.4 EntryOptionPlugin.js

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;
```

10.5 Compilation.js

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.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)
    _addModuleChain(context,entry,name,async,callback){
        this.createModule({
            name,//所属的代码块的名称 main
            context:this.context,//上下文
            rawRequest:entry,// ./src/index.js
            resource:path.posix.join(context,entry),//此模块entry的的绝对路径
            parser,
            async
        },module=>{this.entries.push(module)},callback);
    }
    createModule(data,addEntry,callback){
        //先创建模块工厂
        const moduleFactory = new NormalModuleFactory();
        let module = moduleFactory.create(data);
        //非常非常重要 模块的ID如何生成? 模块的ID是一个相对于根目录的相对路径
        //index.js ./src/index.js title.js ./src/title.js
        //relative返回一个相对路径 从根目录出出到模块的绝地路径 得到一个相对路径
        module.moduleId = '.'+path.posix.sep+path.posix.relative(this.context,module.resource);
        addEntry&&addEntry(module);
        this.modules.push(module);//把模块添加到完整的模块数组中
        const afterBuild = (err,module)=>{
            if(module.dependencies){//如果一个模块编译完成,发现它有依赖的模块,那么递归编译它的依赖模块
                this.processModuleDependencies(module,(err)=>{
                    //当这个入口模块和它依赖的模块都编译完成了,才会让调用入口模块的回调
                    callback(err,module);
                });
            }else{
                callback(err,module);
            }
        }
        this.buildModule(module,afterBuild);
    }
    processModuleDependencies(module,callback){
        let dependencies= module.dependencies;
        //因为我希望可以并行的同时开始编译依赖的模块,然后等所有依赖的模块全部编译完成后才结束
        async.forEach(dependencies,(dependency,done)=>{
            let {name,context,rawRequest,resource,moduleId} = dependency;
            this.createModule({
                name,
                context,
                rawRequest,
                resource,
                moduleId,
                parser
            },null,done);
```

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

10.6 mainDeferTemplate.ejs

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

11.1 webpack.config.js

```
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'
    }
}
```

11.2 src\index.js

src\index.js

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

11.3 NormalModule.js

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||('./'+path.posix.relative(context,resource));
        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;
```

11.4 less-loader.js

loaders\less-loader.js

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

11.5 style-loader.js

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;
}
```

11.6 index.less

src\index.less

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

11.7 loader-runner.js

webpack\loader-runner.js

```
const fs = require('fs');
const path = require('path');
const readFile = fs.readFile.bind(fs);
const PATH_QUERY_FRAGMENT_REGEXP = /^([^?#]*)(\?[^#]*)?(#.*)?$/;

function parsePathQueryFragment(resource){
  let result = PATH_QUERY_FRAGMENT_REGEXP.exec(resource);
  return {
      path:result[1],
      query:result[2],
      fragment:result[3]
  }
}
function loadLoader(loaderObject){
  let normal = require(loaderObject.path);
  loaderObject.normal = normal;
  loaderObject.pitch = normal.pitch;
  loaderObject.raw = normal.raw;
}
function convertArgs(args,raw){
  if(raw&&!Buffer.isBuffer(args[0])){
    args[0] = Buffer.from(args[0],'utf8');
  }else if(!raw && Buffer.isBuffer(args[0])){
    args[0] = args[0].toString('utf8');
  }
}

function createLoaderObject(loader){
  let obj = {
      path:'',
      query:'',
      fragment:'',
      normal:null,
      pitch:null,
      raw:null,
      data:{},
      pitchExecuted:false,
      normalExecuted:false
  }
  Object.defineProperty(obj,'request',{
      get(){
          return obj.path + obj.query+obj.fragment;
      },
      set(value){
        let splittedRequest = parsePathQueryFragment(value);
        obj.path = splittedRequest.path;
        obj.query = splittedRequest.query;
        obj.fragment = splittedRequest.fragment;
      }
  });
  obj.request = loader;
  return obj;
}
function processResource(options,loaderContext,callback){

    loaderContext.loaderIndex = loaderContext.loaders.length-1;
    let resourcePath = loaderContext.resourcePath;

    options.readResource(resourcePath,function(err,buffer){
        if(err) return callback(error);
        options.resourceBuffer = buffer;
        iterateNormalLoaders(options,loaderContext,[buffer],callback);
    });
}
function iterateNormalLoaders(options,loaderContext,args,callback){
    if(loaderContext.loaderIndex<0){
        return callback(null,args);
    }
    let currentLoaderObject = loaderContext.loaders[loaderContext.loaderIndex];

    if(currentLoaderObject.normalExecuted){
        loaderContext.loaderIndex--;
        return iterateNormalLoaders(options,loaderContext,args,callback)
    }
    let normalFn = currentLoaderObject.normal;
    currentLoaderObject.normalExecuted=true;
    convertArgs(args,currentLoaderObject.raw);
    runSyncOrAsync(normalFn,loaderContext,args,function(err){
        if(err) return callback(err);
        let args = Array.prototype.slice.call(arguments,1);
        iterateNormalLoaders(options,loaderContext,args,callback);
    });
}
function iteratePitchingLoaders(options,loaderContext,callback){
    if(loaderContext.loaderIndex>=loaderContext.loaders.length){
        return processResource(options,loaderContext,callback);
    }

   let currentLoaderObject = loaderContext.loaders[loaderContext.loaderIndex];
   if(currentLoaderObject.pitchExecuted){
    loaderContext.loaderIndex++;
    return iteratePitchingLoaders(options,loaderContext,callback)
   }
   loadLoader(currentLoaderObject);
   let pitchFunction = currentLoaderObject.pitch;
```

```
  currentLoaderObject.pitchExecuted = true;
  if(!pitchFunction){
   return iteratePitchingLoaders(options,loaderContext,callback)
  }
  runSyncOrAsync(
   pitchFunction,
   loaderContext,

   [loaderContext.remainingRequest,loaderContext.previousRequest,loaderContext.data={}],
   function(err,...args){
       if(args.length>0){
           loaderContext.loaderIndex--;
           iterateNormalLoaders(options,loaderContext,args,callback);
       }else{
           iteratePitchingLoaders(options,loaderContext,callback)
       }
   }
  );
}
function runSyncOrAsync(fn,context,args,callback){
  let isSync = true;
  let isDone = false;

  context.async = function(){
     isSync = false;
     return innerCallback;
  }
  const innerCallback = context.callback = function(){
      isDone = true;
      isSync=false;
      callback.apply(null,arguments);
  }

  let result = fn.apply(context,args);

  if(isSync){
      isDone = true;
      return callback(null,result);
  }
}
exports.runLoaders = function(options,callback){

 let resource = options.resource||'';

 let loaders = options.loaders ||[];

 let loaderContext = {};

 let readResource = options.readResource|| readFile;
 let splittedResource = parsePathQueryFragment(resource);
 let resourcePath = splittedResource.path;
 let resourceQuery = splittedResource.query;
 let resourceFragment = splittedResource.fragment;
 let contextDirectory = path.dirname(resourcePath);

 loaders=loaders.map(createLoaderObject);

 loaderContext.context = contextDirectory;
 loaderContext.loaderIndex = 0;
 loaderContext.loaders = loaders;
 loaderContext.resourcePath = resourcePath;
 loaderContext.resourceQuery = resourceQuery;
 loaderContext.resourceFragment = resourceFragment;
 loaderContext.async = null;
 loaderContext.callback = null;

 Object.defineProperty(loaderContext,'resource',{
     get(){
         return loaderContext.resourcePath+loaderContext.resourceQuery+loaderContext.resourceFragment;
     }
 });

 Object.defineProperty(loaderContext,'request',{
   get(){
       return loaderContext.loaders.map(l=>l.request).concat(loaderContext.resource).join('!')
   }
 });

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

 Object.defineProperty(loaderContext,'currentRequest',{
   get(){
       return loaderContext.loaders.slice(loaderContext.loaderIndex).map(l=>l.request).concat(loaderContext.resource).join('!')
   }
 });

 Object.defineProperty(loaderContext,'previousRequest',{
   get(){
       return loaderContext.loaders.slice(0,loaderContext.loaderIndex).map(l=>l.request)
   }
 });

 Object.defineProperty(loaderContext,'query',{
   get(){
      let loader = loaderContext.loaders[loaderContext.loaderIndex];
      return loader.options||loader.query;
   }
 });

 Object.defineProperty(loaderContext,'data',{
```

```
    get(){
       let loader = loaderContext.loaders[loaderContext.loaderIndex];
       return loader.data;
    }
  });
  let processOptions = {
      resourceBuffer :null,
      readResource
  }
  iteratePitchingLoaders(processOptions,loaderContext,function(err,result){
    if(err){
        return callback(err,{});
    }
    callback(null,{
        result,
        resourceBuffer:processOptions.resourceBuffer
    });
  });
}
```