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=94 sentences=565, words=3184

## 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'
    }
}
```

```
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
    })
  );
});
```

src\index.js

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

src\title.js

dist\main.js

```
(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. 创建Compiler.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());

  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);
    }
  }
  compiler.hooks.environment.call();
  compiler.hooks.afterEnvironment.call();
  new WebpackOptionsApply().process(options, compiler);
  return compiler;
}

module.exports = webpack;
```

webpack\Compiler.js

```
const {
  Tapable,
  SyncHook,
  SyncBailHook,
  AsyncSeriesHook,
  AsyncParallelHook
} = require("tapable");
class Compiler extends Tapable {
  constructor(context) {
    super();
    this.hooks = {
      environment: new SyncHook([]),
      afterEnvironment: new SyncHook([]),
      afterPlugins: new SyncHook(["compiler"])
    };
    this.options = {};
    this.context = context;
  }
  run() {
    console.log("开始编译");
  }
}
module.exports = Compiler;
```

webpack\plugins\NodeEnvironmentPlugin.js

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

webpack\WebpackOptionsApply.js

```
module.exports = class WebpackOptionsApply {
  process(options, compiler) {

    compiler.hooks.afterPlugins.call(compiler);
  }
};
```

### 3. 监听make事件

webpack\WebpackOptionsApply.js

```
+const EntryOptionPlugin = require("./plugins/EntryOptionPlugin");
module.exports = class WebpackOptionsApply {
  process(options, compiler) {
     //挂载入口文件插件
+    new EntryOptionPlugin().apply(compiler);
    //触发entryOption事件执行
+    compiler.hooks.entryOption.call(options.context, options.entry);
    //插件绑定结束
    compiler.hooks.afterPlugins.call(compiler);
  }
};
```

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

```
module.exports = 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);
      }
    );
  }
};
```

### 4. make

```
+const {Tapable,SyncHook,SyncBailHook,AsyncSeriesHook,AsyncParallelHook} = require("tapable");
+const Compilation  = require('./Compilation');
class Compiler extends Tapable {
  constructor(context) {
    super();
    this.hooks = {
      environment: new SyncHook([]),
      afterEnvironment: new SyncHook([]),
      afterPlugins: new SyncHook(["compiler"]),
      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"])
    };
    this.options = {};
    this.context = context; //设置上下文路径
  }

+  run(finalCallback) {
+    //编译完成后的回调
+    const onCompiled = (err, compilation) => {
+
+    };
+    //准备运行编译
+    this.hooks.beforeRun.callAsync(this, err => {
+      //运行
+      this.hooks.run.callAsync(this, err => {
+        //开始编译,编译完成后执行conCompiled回调
+        this.compile(onCompiled);
+      });
+    });
+  }
+  newCompilation(params){
+    const compilation = new Compilation(this);
+    this.hooks.thisCompilation.call(compilation, params);
+      this.hooks.compilation.call(compilation, params);
+    return compilation;
+  }
+  compile(onCompiled){
+    this.hooks.beforeCompile.callAsync({}, err => {
+      this.hooks.compile.call();
+      const compilation = this.newCompilation();
+      this.hooks.make.callAsync(compilation, err => {
+          console.log(err,'make完成')
+      });
+    });
+  }
}
module.exports = Compiler;
```

webpack\Compilation.js

```
const normalModuleFactory = require('./NormalModuleFactory');
const {Tapable,SyncHook,SyncBailHook,AsyncSeriesHook,AsyncParallelHook} = require("tapable");
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.hooks = {
          addEntry: new SyncHook(["entry", "name"])
        }
        this.entries=[];
    }

    addEntry(context, entry, name, finallyCallback) {
      this.hooks.addEntry.call(entry, name);
      this._addModuleChain(context,entry,name);
      finallyCallback();
    }
    _addModuleChain(context,entry,name){
      let module = normalModuleFactory.create(
          {name,
          context:this.context,
          request:path.posix.join(context,entry)});
      module.build(this);
      this.entries.push(module);
    }
}
module.exports = Compilation;
```

webpack\NormalModuleFactory.js

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

webpack\NormalModule.js

```
class NormalModule{
    constructor({name,context,request}){
      this.name = name;
      this.context = context;
      this.request = request;
    }
    build(compilation){
        console.log('开始编译入口模块');
    }
}
module.exports = NormalModule;
```

### 5. build

webpack\Compilation.js

```
const normalModuleFactory = require('./NormalModuleFactory');
const {Tapable,SyncHook,SyncBailHook,AsyncSeriesHook,AsyncParallelHook} = require("tapable");
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.hooks = {
          addEntry: new SyncHook(["entry", "name"])
        }
+        this.entries=[];
+        this._modules = {}; //模块代码
+        this.modules=[];
    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, finallyCallback) {
      this.hooks.addEntry.call(entry, name);//开始增加入口
      this._addModuleChain(context,entry,name);
+      console.log('编译完成');
+      console.log(this);
      finallyCallback();
   }
   //增加模块链
   _addModuleChain(context,entry,name){
     let module = normalModuleFactory.create(
         {name,  //模块所属的代码块的名称
         context:this.context,//上下文
         request:path.posix.join(context,entry)});//模块完整路径
     module.build(this);//开始编译模块
     this.entries.push(module);//把编译好的模块添加到入口列表里面
   }
   //编译依赖的模块
+   buildDependencies(module,dependencies){
+     module.dependencies = dependencies.map(data =>{//映射老模块到新的模块
+        let module = normalModuleFactory.create(data);//创建新的模块
+        return module.build(this);//编译模块并返回自己
+     });
+   }
}
module.exports = Compilation;
```

webpack\NormalModule.js

```
+const fs = require('fs');
+const ejs = require('ejs');
+const path = require('path');
+const babylon = require('babylon');
+const t = require('babel-types');
+const generate = require('babel-generator').default;
+const traverse = require('babel-traverse').default;
class NormalModule{
    constructor({name,context,request}){
      this.name = name;
      this.context = context;
      this.request = request;
+      this.dependencies = [];
+      this.moduleId;
+      this._ast;
+      this._source;
    }
    build(compilation){
+        let originalSource = compilation.inputFileSystem.readFileSync(this.request,'utf8');
+        const ast = babylon.parse(originalSource);
+        let dependencies = [];
+        traverse(ast,{
+            CallExpression:(nodePath)=>{
+                if (nodePath.node.callee.name == 'require') {
+                    //获取当前节点
+                    let node = nodePath.node;
+                    //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 extension = moduleName.split(path.posix.sep).pop().indexOf('.')==-1?'.js':'';
+                    //获取依赖模块的绝对路径
+                    let dependencyRequest = path.posix.join(path.posix.dirname(this.request),moduleName+extension);
+                    //获取依赖模块的模块ID
+                    let dependencyModuleId = './'+path.posix.relative(this.context,dependencyRequest);
+                    //把依赖对象添加到依赖列表里
+                    dependencies.push({name:this.name,context:this.context,request:dependencyRequest});
+                    //修改加载的模块ID名称
+                    node.arguments = [t.stringLiteral(dependencyModuleId)];
+                }
+            }
+        });
+        //生成新的代码
+        let {code} = generate(ast);
+        //获取模块的来源代码
+        this._source = code;
+        //获得语法树
+        this._ast = ast;
+        //获取模块ID
+        this.moduleId = './'+path.posix.relative(this.context,this.request);
+        //添加到模块数组里
+        compilation.modules.push(this);
+        //KEY为模块的绝对路径 值为模块转译后的代码
+        compilation._modules[this.request] = code;
+        //编译依赖项
+        compilation.buildDependencies(this,dependencies);
+        return this;
    }
}
module.exports = NormalModule;
```

### 6. seal封装chunk

webpack\Compiler.js

```
this.hooks = {
      compilation: new SyncHook(["compilation", "params"]),
+      afterCompile:new SyncHook(["params"]),
      done: new AsyncSeriesHook(["stats"])
    };

compile(onCompiled){
    this.hooks.beforeCompile.callAsync({}, err => {
      this.hooks.compile.call();
      const compilation = this.newCompilation();
      this.hooks.make.callAsync(compilation, err => {
+         compilation.seal(err => {
+           this.hooks.afterCompile.callAsync(compilation, err => {
+              return onCompiled(null, compilation);
+           });
+         });
      });
    });
  }
```

webpack\Compilation.js

```
+ let Chunk = require('./Chunk');
class Compilation extends Tapable {
    constructor(compiler) {
        super();
        this.hooks = {
          addEntry: new SyncHook(["entry", "name"]),
+          seal: new SyncHook([]),
+                beforeChunks: new SyncHook([]),
+                afterChunks: new SyncHook(["chunks"])
        }
        this.entries=[];    //入口模块
        this._modules = {}; //模块代码
        this.modules=[];    //所有模块
+        this.chunks = [];   //代码块
    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, finallyCallback) {
      this.hooks.addEntry.call(entry, name);//开始增加入口
      this._addModuleChain(context,entry,name);
      finallyCallback();
   }

+  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
+  }
}
module.exports = Compilation;
```

### 7. emit

webpack\Compiler.js

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

webpack\Compilation.js

```
const normalModuleFactory = require('./NormalModuleFactory');
const {Tapable,SyncHook,SyncBailHook,AsyncSeriesHook,AsyncParallelHook} = require("tapable");
const path = require('path');
+ const Chunk = require('./Chunk');
+ const fs = require('fs');
+ const ejs = require('ejs');
+ const mainTemplate = fs.readFileSync(path.join(__dirname, '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.hooks = {
          addEntry: new SyncHook(["entry", "name"]),
          seal: new SyncHook([]),
                beforeChunks: new SyncHook([]),
                afterChunks: new SyncHook(["chunks"])
        }
        this.entries=[];    //入口模块
        this._modules = {}; //模块代码
        this.modules=[];    //所有模块
        this.chunks = [];   //代码块
+       this.files=[];
+       this.assets = {};   //资源
    }
    //context ./src/index.js main callback(终级回调)
    addEntry(context, entry, name, finallyCallback) {
      this.hooks.addEntry.call(entry, name);//开始增加入口
      this._addModuleChain(context,entry,name);
      finallyCallback();
    }
    //增加模块链
    _addModuleChain(context,entry,name){
      let module = normalModuleFactory.create(
          {name,  //模块所属的代码块的名称
          context:this.context,//上下文
          request:path.posix.join(context,entry)});//模块完整路径
     module.build(this);//开始编译模块
     this.entries.push(module);//把编译好的模块添加到入口列表里面
    }
    //编译依赖的模块
    buildDependencies(module,dependencies){
     module.dependencies = dependencies.map(data =>{//映射老模块到新的模块
        let module = normalModuleFactory.create(data);//创建新的模块
        return module.build(this);//编译模块并返回自己
     });
  }
  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);
+   }
}
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__) {

              }),

    });
```

webpack\Stats.js

```
class Stats{
    constructor(compilation){
        this.files = compilation.files;
        this.modules = compilation.modules;
        this.chunks = compilation.chunks;
    }
}
module.exports = Stats;
```

## 8. 支持loader

webpack\NormalModule.js

```
const fs = require('fs');
const ejs = require('ejs');
const path = require('path');
const babylon = require('babylon');
const t = require('babel-types');
const generate = require('babel-generator').default;
const traverse = require('babel-traverse').default;
class NormalModule{
    constructor({name,context,request}){
      this.name = name;
      this.context = context;
      this.request = request;
      this.dependencies = [];
      this.moduleId;
      this._ast;
      this._source;
    }
+    getSource(request,compilation){
+        let source = compilation.inputFileSystem.readFileSync(this.request,'utf8');
+        let { module: { rules } } = compilation.options;
+        for (let i = 0; i < rules.length; i++) {
+            let rule = rules[i];
+            if (rule.test.test(request)) {
+                let loaders = rule.use;
+                let loaderIndex = loaders.length - 1;
+                let iterateLoaders = ()=>{
+                    let loaderName = loaders[loaderIndex];
+                    let loader = require(path.resolve(this.context, 'loaders', loaderName));
+                    source = loader(source);
+                    if (loaderIndex > 0) {
+                        loaderIndex--;
+                        iterateLoaders();
+                    }
+                }
+                iterateLoaders();
+                break;
+            }
+        }
+        return source;
+    }
    build(compilation){
+        let originalSource = this.getSource(this.request,compilation);
        const ast = babylon.parse(originalSource);
        let dependencies = [];
        traverse(ast,{
            CallExpression:(nodePath)=>{
                if (nodePath.node.callee.name == 'require') {
                    //获取当前节点
                    let node = nodePath.node;
                    //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 extension = moduleName.split(path.posix.sep).pop().indexOf('.')==-1?'.js':'';
                    //获取依赖模块的绝对路径
                    let dependencyRequest = path.posix.join(path.posix.dirname(this.request),moduleName+extension);
                    //获取依赖模块的模块ID
                    let dependencyModuleId = './'+path.posix.relative(this.context,dependencyRequest);
                    //把依赖对象添加到依赖列表里
                    dependencies.push({name:this.name,context:this.context,request:dependencyRequest});
                    //修改加载的模块ID名称
                    node.arguments = [t.stringLiteral(dependencyModuleId)];
                }
            }
        });
        //生成新的代码
        let {code} = generate(ast);
        //获取模块的来源代码
        this._source = code;
        //获得语法树
        this._ast = ast;
        //获取模块ID
        this.moduleId = './'+path.posix.relative(this.context,this.request);
        //添加到模块数组里
        compilation.modules.push(this);
        //KEY为模块的绝对路径 值为模块转译后的代码
        compilation._modules[this.request] = code;
        //编译依赖项
        compilation.buildDependencies(this,dependencies);
        return this;
    }
}
module.exports = NormalModule;
```

src\index.js

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

index.less

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

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;
}
```