**代码分离**

此特性能够把代码分离到不同的 bundle 中，然后可以按需加载或并行加载这些文件。意思就是动态加载

有三种常用的代码分离方法：

入口起点：使用 [entry](https://www.webpackjs.com/configuration/entry-context) 配置手动地分离代码。

防止重复：使用 [CommonsChunkPlugin](https://www.webpackjs.com/plugins/commons-chunk-plugin) 去重和分离 chunk。

动态导入（推荐）：通过模块的内联函数调用来分离代码。

**入口起点(entry points)**

先来看看如何从 main bundle 中分离另一个模块：

webpack.config.js

const path = require('path');

const HTMLWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: {

index: './src/index.js',

another: './src/another-module.js'

},

plugins: [

new HTMLWebpackPlugin({

title: 'Code Splitting'

})

],

output: {

filename: '[name].bundle.js',

path: path.resolve(\_\_dirname, 'dist')

}

};

这将生成如下构建结果：

Hash: 309402710a14167f42a8

Version: webpack 2.6.1

Time: 570ms

Asset Size Chunks Chunk Names

index.bundle.js 544 kB 0 [emitted] [big] index

another.bundle.js 544 kB 1 [emitted] [big] another

[0] ./~/lodash/lodash.js 540 kB {0} {1} [built]

[1] (webpack)/buildin/global.js 509 bytes {0} {1} [built]

[2] (webpack)/buildin/module.js 517 bytes {0} {1} [built]

[3] ./src/another-module.js 87 bytes {1} [built]

[4] ./src/index.js 216 bytes {0} [built]

正如前面提到的，这种方法存在一些问题:

如果入口 chunks 之间包含重复的模块，那些重复模块都会被引入到各个 bundle 中。

这种方法不够灵活，并且不能将核心应用程序逻辑进行动态拆分代码。

**防止重复(prevent duplication)**

[CommonsChunkPlugin](https://www.webpackjs.com/plugins/commons-chunk-plugin) 插件可以将公共的依赖模块提取到已有的入口 chunk 中，或者提取到一个新生成的 chunk。

webpack.config.js

const path = require('path');

+ const webpack = require('webpack');

const HTMLWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: {

index: './src/index.js',

another: './src/another-module.js'

},

plugins: [

new HTMLWebpackPlugin({

title: 'Code Splitting'

- })

+ }),

+ new webpack.optimize.CommonsChunkPlugin({

+ name: 'common' // 指定公共 bundle 的名称。

+ })

],

output: {

filename: '[name].bundle.js',

path: path.resolve(\_\_dirname, 'dist')

}

};

这里我们使用 [CommonsChunkPlugin](https://www.webpackjs.com/plugins/commons-chunk-plugin) 之后，现在应该可以看出，index.bundle.js 中已经移除了重复的依赖模块。

Hash: 70a59f8d46ff12575481

Version: webpack 2.6.1

Time: 510ms

Asset Size Chunks Chunk Names

index.bundle.js 665 bytes 0 [emitted] index

another.bundle.js 537 bytes 1 [emitted] another

common.bundle.js 547 kB 2 [emitted] [big] common

[0] ./~/lodash/lodash.js 540 kB {2} [built]

[1] (webpack)/buildin/global.js 509 bytes {2} [built]

[2] (webpack)/buildin/module.js 517 bytes {2} [built]

[3] ./src/another-module.js 87 bytes {1} [built]

[4] ./src/index.js 216 bytes {0} [built]

**动态导入(dynamic imports)**

webpack.config.js

const path = require('path');

const HTMLWebpackPlugin = require('html-webpack-plugin');

module.exports = {

entry: {

+ index: './src/index.js'

},

plugins: [

new HTMLWebpackPlugin({

title: 'Code Splitting'

+ })

],

output: {

filename: '[name].bundle.js',

+ chunkFilename: '[name].bundle.js',

path: path.resolve(\_\_dirname, 'dist')

}

};

注意，这里使用了 chunkFilename，它决定非入口 chunk 的名称。接着，更新我们的项目，移除掉那些现在不会用到的文件:

**src/index.js**

function getComponent() {

    // 当 getComponent 运行时，动态加载 lodash.js

    // 注意，下面的 /\* webpackChunkName: "lodash" \*/ 不是注释

    return import(/\* webpackChunkName: "lodash" \*/ 'lodash').then(\_ => {

        var element = document.createElement('div');

        element.innerHTML = \_.join(['Hello', 'webpack'], ' ');

        return element;

    }).catch(error => 'An error occurred while loading the component');

}

getComponent().then(component => {

    document.body.appendChild(component);

})

注意，在注释中使用了 webpackChunkName。这样做会导致我们的 bundle 被命名为 lodash.bundle.js ，而不是 [id].bundle.js 。让我们执行 webpack，查看 lodash 是否会分离到一个单独的 bundle：

Hash: a27e5bf1dd73c675d5c9

Version: webpack 2.6.1

Time: 544ms

Asset Size Chunks Chunk Names

lodash.bundle.js 541 kB 0 [emitted] [big] lodash

index.bundle.js 6.35 kB 1 [emitted] index

[0] ./~/lodash/lodash.js 540 kB {0} [built]

[1] ./src/index.js 377 bytes {1} [built]

[2] (webpack)/buildin/global.js 509 bytes {0} [built]

[3] (webpack)/buildin/module.js 517 bytes {0} [built]

由于 import() 会返回一个 promise，因此它可以和 [async 函数](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function)一起使用。下面是如何通过 async 函数简化代码：

**src/index.js**

async function getComponent() {

    var element = document.createElement('div');

    const \_ = await import(/\* webpackChunkName: "lodash" \*/ 'lodash');

element.innerHTML = \_.join(['Hello', 'webpack'], ' ');

    return element;

}