---

link: null
title: 珠峰架构师成长计划
description: https://static.zhufengpeixun.com/http_xie_yi_rfc2616_zhong_ying_wen_shuang_ban_1637749432228.zip
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=97 sentences=182, words=1012

---

1. 代码分割

- 对于大的Web应用来讲，将所有的代码都放在一个文件中显然是不够有效的，特别是当你的某些代码块是在某些特殊的时候才会被用到。
- webpack有一个功能就是将你的代码库分割成chunks语块，当代码运行到需要它们的时候再进行加载

2. 入口点分割

- Entry Points：入口文件设置的时候可以配置
- 这种方法的问题
    - 如果入口 chunks 之间包含重复的模块(lodash)，那些重复模块都会被引入到各个 bundle 中
    - 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

```
{
  entry: {
   page1: "./src/page1.js",
   page2: "./src/page2.js"
  }
}
```

https://static.zhufengpeixun.com/http_xie_yi_rfc2616_zhong_ying_wen_shuang_ban_1637749432228.zip
(https://static.zhufengpeixun.com/http_xie_yi_rfc2616_zhong_ying_wen_shuang_ban_1637749432228.zip)

3 动态导入和懒加载

- 用户当前需要用什么功能就只加载这个功能对应的代码，也就是所谓的按需加载 在给单页应用做按需加载优化时
- 一般采用以下原则：
    - 对网站功能进行划分，每一类一个chunk
    - 对于首次打开页面需要的功能直接加载，尽快展示给用户,某些依赖大量代码的功能点可以按需加载
    - 被分割出去的代码需要一个按需加载的时机

3.1 video.js

hello.js

```
module.exports = "video";
```

index.js

```
document.querySelector('#play').addEventListener('click',() => {
    import('./video').then(result => {
        console.log(result.default);
    });
});
```

index.html

```
<button id="play">播放button>
```

3.2 prefetch(预先拉取)

- prefetch 跟 preload 不同，它的作用是告诉浏览器未来可能会使用到的某个资源，浏览器就会在闲时去加载对应的资源，若能预测到用户的行为，比如懒加载，点击到其它页面等则相当于提前预加载了需要的资源

```
<link rel="prefetch" href="utils.js" as="script">
```

```
button.addEventListener('click', () => {
  import(
    `./utils.js`

  ).then(result => {
    result.default.log('hello');
  })
});
```

3.3 preload(预先加载)

- preload通常用于本页面要用到的关键资源，包括关键js、字体、css文件
- preload将会把资源得下载顺序权重提高，使得关键数据提前下载好,优化页面打开速度
- 在资源上添加预先加载的注释，你指明该模块需要立即被使用
- 一个资源的加载的优先级被分为五个级别,分别是
    - Highest 最高
    - High 高
    - Medium 中等
    - Low 低
    - Lowest 最低
- 异步/延迟/插入的脚本（无论在什么位置）在网络优先级中是 Low
- link-rel-prefetch-preload-in-webpack (https://medium.com/webpack/link-rel-prefetch-preload-in-webpack-51a52358f84c)
- Support for webpackPrefetch and webpackPreload (https://github.com/jantimon/html-webpack-plugin/issues/1317)
- preload-webpack-plugin (https://www.npmjs.com/package/@vue/preload-webpack-plugin)
- webpackpreload-webpack-plugin (https://www.npmjs.com/package/webpackpreload-webpack-plugin)
- ImportPlugin.js (https://github.com/webpack/webpack/blob/c181294865dca01b28e6e316636fef5f2aad4eb6/lib/dependencies/ImportParserPlugin.js#L108-L121)

```
$ npm install --save-dev webpackpreload-webpack-plugin
```

□

```
"preload" as="script" href="utils.js">
```

```
import(
  `./video.js`

)
```

webpackpreload-webpack-plugin

```
const HtmlWebpackPlugin = require("html-webpack-plugin");
class WebpackpreloadWebpackPlugin {
  apply(compiler) {
    compiler.hooks.compilation.tap('PreloadWebpackPlugin', (compilation) => {
      HtmlWebpackPlugin.getHooks(compilation).alterAssetTags.tap('PreloadWebpackPlugin', (htmlData) => {
        const { publicPath, assetTags } = htmlData;
        const { entrypoints, moduleGraph, chunkGraph } = compilation;
        for (const entrypoint of entrypoints) {
          const preloaded = entrypoint[1].getChildrenByOrders(moduleGraph, chunkGraph).preload;
          if (!preloaded) return;
          const chunks = new Set();
          for (const group of preloaded) {
            for (const chunk of group.chunks) chunks.add(chunk);
          }
          const files = new Set()
          for (const chunk of chunks) {
            for (const file of chunk.files) files.add(file);
          }
          const links = [];
          for (const file of files) {
            links.push({
              tagName: 'link',
              attributes: {
                rel: 'preload',
                href: `${publicPath}${file}`
              }
            });
          }
          assetTags.styles.unshift(...links);
        }
      });
    });
  }
}
module.exports = WebpackpreloadWebpackPlugin;
```

plugins\ImportPlugin.js

```
class ImportPlugin {
    apply(compiler) {
        compiler.hooks.compilation.tap(
            "ImportPlugin",
            (compilation, { normalModuleFactory }) => {
                normalModuleFactory.hooks.parser
                    .for("javascript/auto")
                    .tap("ImportPlugin", (parser) => {
                        parser.hooks.importCall.tap("ImportParserPlugin", expr => {
                            const { options } = parser.parseCommentOptions(expr.range);
                            console.log(options);
                        });
                    });
            }
        );
    }
}
module.exports = ImportPlugin;
```

3.4 preload vs prefetch

- preload 是告诉浏览器页面必定需要的资源，浏览器一定会加载这些资源
- 而 prefetch 是告诉浏览器页面可能需要的资源，浏览器不一定会加载这些资源
- 所以建议：对于当前页面很有必要的资源使用 preload,对于可能在将来的页面中使用的资源使用 prefetch

4. 提取公共代码

- split-chunks-plugin (https://webpack.js.org/plugins/split-chunks-plugin)
- split-chunks-plugin (https://webpack.docschina.org/plugins/split-chunks-plugin/)
- common-chunk-and-vendor-chunk (https://github.com/webpack/webpack/tree/master/examples/common-chunk-and-vendor-chunk)
- 怎么配置单页应用?怎么配置多页应用?

4.1 为什么需要提取公共代码

- 大网站有多个页面，每个页面由于采用相同技术栈和样式代码，会包含很多公共代码，如果都包含进来会有问题
- 相同的资源被重复的加载，浪费用户的流量和服务器的成本；
- 每个页面需要加载的资源太大，导致网页首屏加载缓慢，影响用户体验。
- 如果能把公共代码抽离成单独文件进行加载能进行优化，可以减少网络传输流量，降低服务器成本

4.2 如何提取

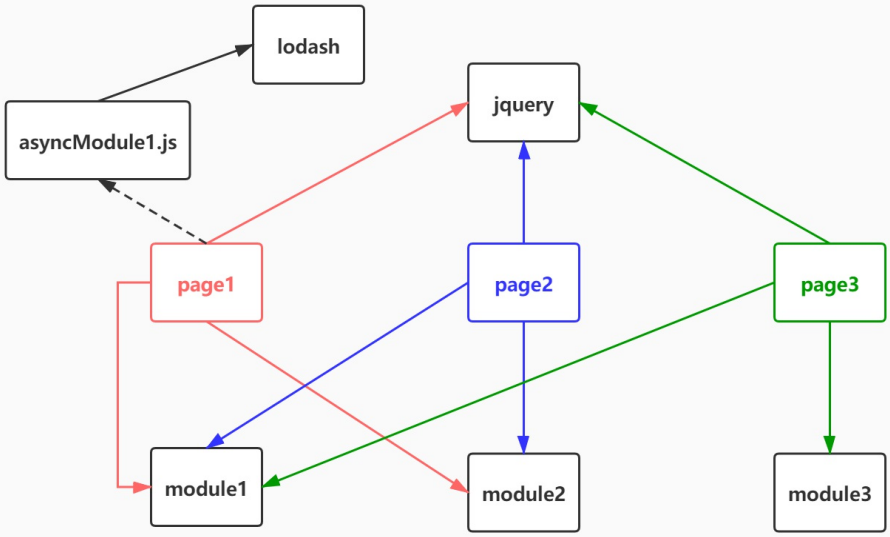
- 基础类库，方便长期缓存
- 页面之间的公用代码
- 各个页面单独生成文件

4.3 module chunk bundle

- module：就是js的模块化webpack支持commonJS、ES6等模块化规范，简单来说就是你通过import语句引入的代码
- chunk: chunk是webpack根据功能拆分出来的，包含三种情况
    - 你的项目入口（entry）
    - 通过import()动态引入的代码
    - 通过splitChunks拆分出来的代码
- bundle：bundle是webpack打包之后的各个文件，一般就是和chunk是一对一的关系，bundle就是对chunk进行编译压缩打包等处理之后的产出

4.4 splitChunks

- split-chunks-plugin (https://webpack.js.org/plugins/split-chunks-plugin)
- 将optimization.runtimeChunk (https://webpack.js.org/configuration/optimization/#optimizationruntimechunk)设置为 true 或 'multiple'，会为每个入口添加一个只含有 runtime 的额外 chunk




```
                         Asset       Size                          Chunks             Chunk Names
          asyncModule1.chunk.js  740 bytes                asyncModule1  [emitted]  asyncModule1
                    index.html  498 bytes                              [emitted]
                       page1.js   10.6 KiB                       page1  [emitted]  page1
          page1~page2.chunk.js  302 bytes              page1~page2  [emitted]  page1~page2
    page1~page2~page3.chunk.js  308 bytes        page1~page2~page3  [emitted]  page1~page2~page3
                       page2.js   7.52 KiB                       page2  [emitted]  page2
                       page3.js   7.72 KiB                       page3  [emitted]  page3
  vendors~asyncModule1.chunk.js    532 KiB       vendors~asyncModule1  [emitted]  vendors~asyncModule1
vendors~page1~page2~page3.chunk.js    282 KiB  vendors~page1~page2~page3  [emitted]  vendors~page1~page2~page3
Entrypoint page1 = vendors~page1~page2~page3.chunk.js page1~page2~page3.chunk.js page1~page2.chunk.js page1.js
Entrypoint page2 = vendors~page1~page2~page3.chunk.js page1~page2~page3.chunk.js page1~page2.chunk.js page2.js
Entrypoint page3 = vendors~page1~page2~page3.chunk.js page1~page2~page3.chunk.js page3.js
```


4.4.1 工作流程

- 1. SplitChunksPlugi先尝试把 `minChunks`规则的模块抽取到单独的 `Chunk`中
- 1. 判断该Chunk是否满足 `maxInitialRequests`配置项的要求
- 1. 判断体积是否满足 `minSize`的大小，如果小于 `minSize`则不分包，如果大于 `minSize`判断是否超过 `maxSize`,如果大于 `maxSize`则继续拆分成更小的包

4.4.1 webpack.config.js

- 请求数是指加载一个 `Chunk`时所需要加载的所有的分包数量,包括 `Initial Chunk`，但不包括 `Async Chunk`和 `runtimeChunk`
- 用于设置 Initial Chunk 最大并行请求数
- 用于设置 Async Chunk 最大并行请求数

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const AssetPlugin = require('./asset-plugin');
module.exports = {
    mode: 'development',
    devtool: false,
    entry: {
        page1: "./src/page1.js",
        page2: "./src/page2.js",
        page3: "./src/page3.js",
    },
    optimization: {
        splitChunks: {

            chunks: 'all',

            minSize: 0,

            minChunks: 1,

            maxAsyncRequests: 3,

            maxInitialRequests: 5,

            automaticNameDelimiter: '~',
            cacheGroups: {
                defaultVendors: {
                    test: /[\\/]node_modules[\\/]/,
                    priority: -10
                },
                default: {
                    minChunks: 2,
                    priority: -20
                },
            },
        },
        runtimeChunk: true
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: './src/index.html',
            chunks: ["page1"],
            filename: 'page1.html'
        }),
        new HtmlWebpackPlugin({
            template: './src/index.html',
            chunks: ["page2"],
            filename: 'page2.html'
        }),
        new HtmlWebpackPlugin({
            template: './src/index.html',
            chunks: ["page3"],
            filename: 'page3.html'
        }),
        new AssetPlugin()
    ]
}
```

4.4.2 webpack-assets-plugin.js

plugins\webpack-assets-plugin.js

```
class WebpackAssetsPlugin {
  constructor(options) {
    this.options = options;
  }
  apply(compiler) {

    compiler.hooks.compilation.tap('WebpackAssetsPlugin', (compilation) => {

      compilation.hooks.chunkAsset.tap('WebpackAssetsPlugin', (chunk, filename) => {
        console.log(chunk.id, filename);
      });
    });
  }
}
module.exports = WebpackAssetsPlugin;
```

4.4.3 page1.js

```
let module1 = require('./module1');
let module2 = require('./module2');
let $ = require('jquery');
console.log(module1,module2,$);
import(  './asyncModule1');
```

4.4.4 page2.js

```
let module1 = require('./module1');
let module2 = require('./module2');
let $ = require('jquery');
console.log(module1,module2,$);
```

4.4.5 page3.js

```
let module1 = require('./module1');
let module3 = require('./module3');
let $ = require('jquery');
console.log(module1,module3,$);
```

4.4.6 module1.js

```
module.exports = 'module1';
```

4.4.7 module2.js

```
console.log("module2");
```

4.4.8 module3.js

```
console.log("module3");
```

4.4.9 asyncModule1.js

```
import _ from 'lodash';
console.log(_);
```

4.4.10 打包后的结果

```

page1.js
page2.js
page3.js

src_asyncModule1_js.js

defaultVendors-node_modules_jquery_dist_jquery_js.js
defaultVendors-node_modules_lodash_lodash_js.js

default-src_module1_js.js
default-src_module2_js.js
```

4.4.11 计算过程

```
let page1Chunk= {
    name:'page1',
    modules:['A','B','C','lodash']
}

let page2Chunk = {
    name:'page2',
    module:['C','D','E','lodash']
}

let  cacheGroups= {
    vendor: {
      test: /lodash/,
    },
    default: {
      minChunks: 2,
    }
};

let vendorChunk = {
    name:`vendor~node_modules_lodash_js`,
    modules:['lodash']
}
let defaultChunk = {
    name:`default~page1~page2`,
    modules:['C']
}
```

4.5 reuseExistingChunk

- reuseExistingChunk (https://webpack.js.org/plugins/split-chunks-plugin/#splitchunkscachegroupscachegroupreuseexistingchunk)表示如果当前的代码包含已经被从主bundle中分割出去的模块，它将会被重用，而不会生成一个新的代码块

4.5.1 index.js

4.5.2 webpack.config.js

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const AssetPlugin = require('./asset-plugin');
module.exports = {
    mode: 'development',
    devtool: false,
+   entry: './src/index.js',
    optimization: {
        splitChunks: {
            // 表示选择哪些 chunks 进行分割，可选值有：async，initial和all
            chunks: 'all',
            // 表示新分离出的chunk必须大于等于minSize，默认为30000，约30kb。
            minSize: 0,//默认值是20000,生成的代码块的最小尺寸
            // 表示一个模块至少应被minChunks个chunk所包含才能分割。默认为1。
            minChunks: 1,
            // 表示按需加载文件时，并行请求的最大数目。默认为5。
            maxAsyncRequests: 3,
            // 表示加载入口文件时，并行请求的最大数目。默认为3
            maxInitialRequests: 5,
            // 表示拆分出的chunk的名称连接符。默认为~。如chunk~vendors.js
            automaticNameDelimiter: '~',
+           cacheGroups: {
+               defaultVendors: false,
+               default: false,
+               common: {
+                   minChunks: 1,
+                   reuseExistingChunk: false
+               }
+           }
        },
+       runtimeChunk: false
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: './src/index.html',
            filename: 'index.html'
        })
        new AssetPlugin()
    ]
}
```

4.5.3 结果

```
main main.js
common-src_index_js common-src_index_js.js

main main.js
```