umi打包优化

目标

1、减少最终web项目体积

2、实现按路由动态引入

3、再次将大文件按规则拆分为独立的小文件

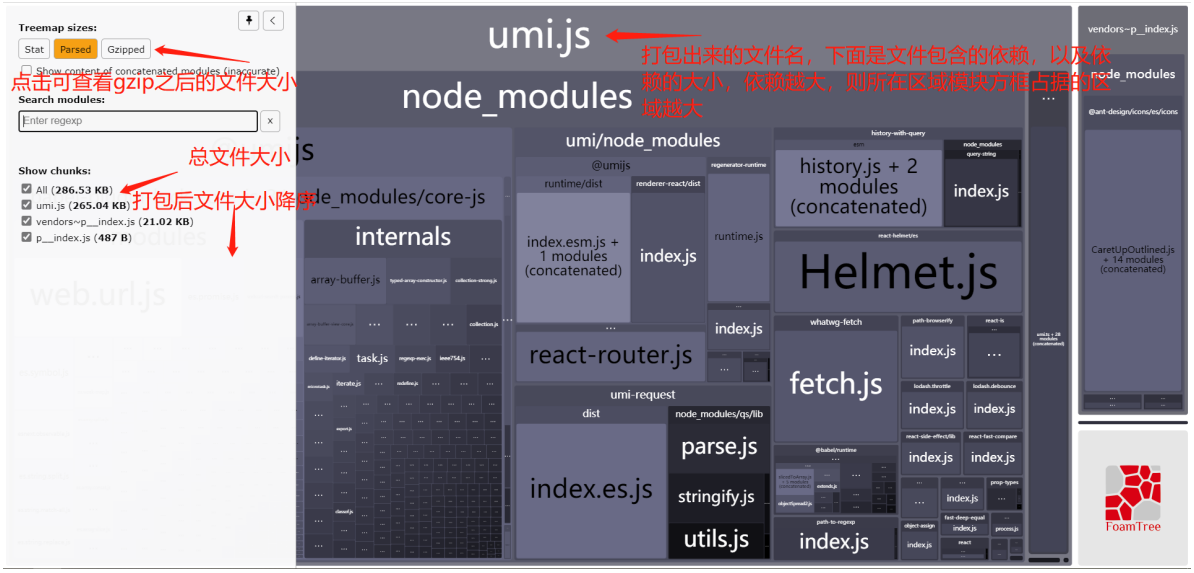
需要使用的工具

`webpack`：打包工具

`webpack-bundle-analyzer`：性能分析工具

以上两个工具`umi`都已经整合好了. 只需直接使用即可.

执行性能分析命令: `cross-env ANALYZE=1 umi build`, 此时能看到如下类似界面



实现：减少最终web项目体积

什么时候适合采用此方式优化?

主要目标客户所在网络，访问web项目所在服务器比较慢，但访问cdn比较快时.

预设情景:

web项目, 打包后, 第三方依赖1000KB, 项目代码: 200KB

客户访问web项目, 服务器资源的网速为: 100KB/s, 而访问cdn的速度为500KB/s.

如果资源全部在服务器, 那么客户完全加载完静态资源需要的时间为: `(1000+200)/100 = 12秒`

如果第三方依赖在cdn, 只有项目代码在web服务器, 那么客户完全加载完静态资源需要的时间为: `1000/500=2秒`, 加载完代码资源需要时间为: `200/500=0.4秒`, 总计: `2.4秒`, 与完全访问服务器比总计能减少`9.6秒`, 快了`5`倍的时间

查看每个第三方依赖的体积, 逐步文件体积大的依赖, 改为cdn方式引入, 而非打包到项目的js中

优化方式:

不断执行如下3步, 直到达到既定目标

1、执行`` `cross-env ANALYZE=1 umi build `` 查看依赖模块以及打包之后的文件大小

2、将体积庞大的第三方依赖, 改为cdn引入

3、启动项目运行下, 看是否有导致项目无法运行的情况

umi配置详见

```
// .umirc.ts

import { defineConfig } from 'umi';

export default defineConfig({
  ... 省略不必要代码 ...
  // umi默认antd引用方式会使用按需引入, 而按需加载会导致, externals的配置方式变复杂, 因此禁用默认antd.
  antd: false,
  // 禁用不必要的其它插件, 减小打包体积
  layout: false,
  // 此处的配置会被转换为一个个的script标签, 用来实现cdn方式引入js
  scripts: [

 'https://cdn.bootcdn.net/ajax/libs/react/17.0.1/umd/react.production.min.js',
    'https://cdn.bootcdn.net/ajax/libs/react-dom/17.0.1/umd/react-dom.production.min.js',
    'https://cdn.bootcdn.net/ajax/libs/antd/4.10.3/antd.min.js',
    'https://cdn.bootcdn.net/ajax/libs/antd/4.10.3/theme.js',
  ],
  // 此处的配置会被转换为一个个的link标签, 用来实现cdn方式引入css
  styles:[
    'https://cdn.bootcdn.net/ajax/libs/antd/4.10.3/antd.min.css',
  ],
  // externals(告诉webpack,在打包时要排除的依赖), 配置方式详见: https://v4.webpack.docschina.org/guides/author-libraries
  // 此处配置的作为为: 打包时,排除react,react-dom,antd包, 并且告诉webpack, cdn方式引入的react会在window对象中放一个React对象, react-dom会在window对象中放一个ReactDOM对象. antd会定义一个全局的antd对象
  externals: {
```

```
    'react': 'window.React',
    'react-dom': 'window.ReactDOM',
    // 只有当antd为全量引入时，该种方式的配置才有效，否则此种方式配置无效
    'antd': 'antd',
  },
  ... 省略不必要代码 ...
});
```

其它引用与参考

[webpack打包优化解决方案]

[配置antd的externals之后，antd还是会被打包进去 [BUG]]

[antd配置了externals，但是如果开启了plugin-layout，还是会打包antd]

[umi externals 还是打包了设置项 比如antd]

[https://v4.webpack.docschina.org/guides/author-libraries]

[webpack打包优化之外部扩展externals的实际应用]

[详解webpack4之splitchunksPlugin代码包分拆]

实现: 按路由动态导入

什么时候适合采用此方式优化?

即使将第三方依赖改为cdn方式引入, 但由于项目代码本身很大, 首次加载依然缓慢的情况.

预设情景:

假设项目由`50`个界面, 打包后的总体积为: 1000KB， 平均每个界面的大小为: `200KB`, 如果客户访问服务器的网速为: `100KB/s`,

那么加载完所有资源需要`10秒`, 而如果只加载`200KB`， 那么只需`2秒`， 减少了`8秒`

优化方式:

[umi配置详见]

1、启用按路由动态引入

```
// .umirc.ts

import { defineConfig } from 'umi';

export default defineConfig({
  ... 省略不必要代码 ...
  dynamicImport: {},
  ... 省略不必要代码 ...
});
```

实现: 再次将大文件按规则拆分为独立的小文件

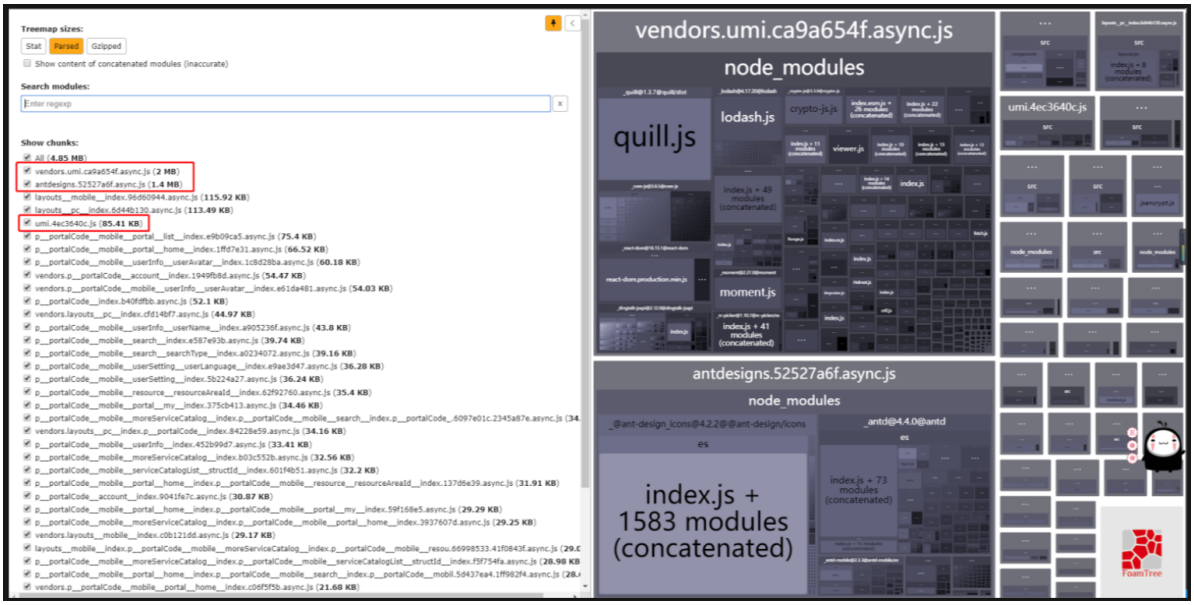
什么时候适合采用此方式优化?

按照前面两步优化之后, 所有的未被排除的第三方依赖被打入了`umi.js`中, 其它业务代码被打入了各自的路由命名文件中. 但是可能某些体积很大的第三方依赖(如:图表库)可能由于某些原因, 并未采用cdn方式引入, 因此也被一同打入`umi.js`中, 但实际首屏加载时可能用不到这些第三方依赖, 此时就应该再次将这些第三方依赖, 按照合适的规则, 打入独立的文件. 只在实际需要使用时, 再动态引入

优化方式:

摘自

```
// .umirc.ts

export default defineConfig({

  ...其他配置
    chainWebpack: chainWebpackConfig(config, { webpack }) {
    // webpack 拆包
    config.optimization.splitChunks({
      chunks: 'all', // 提取 chunks 的时候从哪里提取，如果为 all 那么不管是不是 async 的都可能被抽出 chunk，为 initial 则会从非 async 里面提取。
      automaticNameDelimiter: '.', // 文件名分隔符
      name: true,  // chunk 的名称，如果设置为固定的字符串那么所有的 chunk 都会被合并成一个，这就是为什么 umi 默认只有一个 vendors.async.js。
      minSize: 30000, // byte, == 30 kb，越大那么单个文件越大，chunk 数就会变少（针对于提取公共 chunk 的时候，不管再大也不会把动态加载的模块合并到初始化模块中）当这个值很大的时候就不会做公共部分的抽取了
      maxSize: 0, // 文件的最大尺寸，优先级：maxInitialRequest/maxAsyncRequests < maxSize < minSize，需要注意的是这个如果配置了，umi.js 就可能被拆开，最后构建出来的 chunkMap 中可能就找不到 umi.js 了。
      minChunks: 1, // 被提取的一个模块至少需要在几个 chunk 中被引用，这个值越大，抽取出来的文件就越小
      maxAsyncRequests: 10, // 在做一次按需加载的时候最多有多少个异步请求，为 1 的时候就不会抽取公共 chunk 了
      maxInitialRequests: 5, // 针对一个 entry 做初始化模块分隔的时候的最大文件数，优先级高于 cacheGroup，所以为 1 的时候就不会抽取 initial common 了。
      cacheGroups: {
        antdesigns: { // antdsign
          name: 'antdesigns',
          chunks: 'all',
          test: /(@antd|antd|@ant-design)/,
          priority: 10,
        },
      },
    });
  }
});
```



这一看出多了antdesigns.js文件， 还多了一个vendors.umi.js.

vendors.umi 这个文件从何而来？ 这个是默认拆出来的。

我们只要看到antdesigns.js 就证明我们拆包成功了。

接下来就自己按照自己的项目去拆分包就好了。

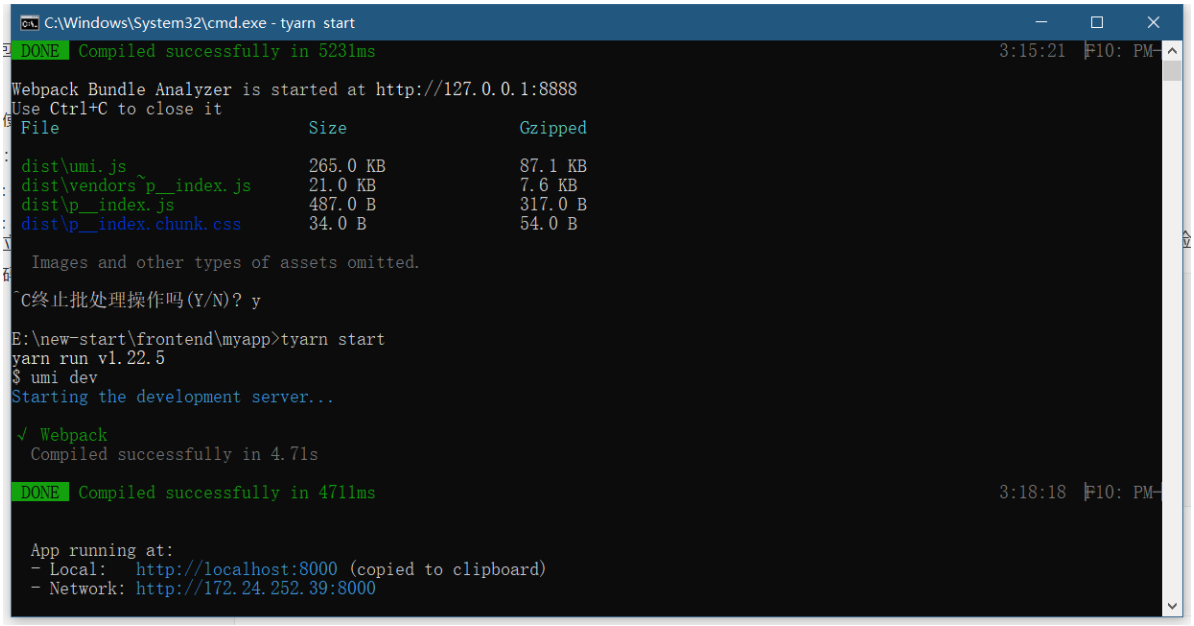
包拆完了，发到线上，发现白屏，打不开，拆出来的js文件都没有去请求，这个时候就应该检查配置，看有没有配置拆出来的包。

```
// .umirc.ts

export default defineConfig({

  ...其他配置

    chunks: ['antdesigns', 'vendors.umi', 'umi']
});
```

源代码

源代码地址

其它


C:\Windows\System32\cmd.exe - tyarn start
DONE Compiled successfully in 5231ms 3:15:21 PM
Webpack Bundle Analyzer is started at http://127.0.0.1:8888
Use Ctrl+C to close it
File Size Gzipped
dist\umi.js 265.0 KB 87.1 KB
dist\vendors~p__index.js 21.0 KB 7.6 KB
dist\p__index.js 487.0 B 317.0 B
dist\p__index.chunk.css 34.0 B 54.0 B
Images and other types of assets omitted.
^C终止批处理操作吗(Y/N)? y
E:\new-start\frontend\myapp>tyarn start
yarn run v1.22.5
$ umi dev
Starting the development server...
√ Webpack
Compiled successfully in 4.71s
DONE Compiled successfully in 4711ms 3:18:18 PM
App running at:
- Local: http://localhost:8000 (copied to clipboard)
- Network: http://172.24.252.39:8000