# 生产环境配置

关注点：

- 如何让客户更快的加载资源
- 如何压缩资源
- 如何添加环境变量优化打包
- 最大限度利用缓存

# 环境配置的封装

如何让Webpack按照不同环境采用不同的配置呢？

- 使用相同的配置文件
  不论什么环境打包都是用webpack.config.js
  在构建开始将环境作为变量传进去
  在webpack.config.js中通过条件判断决定具体使用什么配置

```
// package.json
{
  ...
  "scripts": {
    "dev": "ENV=development webpack-dev-server",
    "build": "ENV=production webpack"
  },
}

// webpack.config.js
const ENV = process.env.ENV;
const isProd = ENV === 'production';
module.exports = {
  output: {
    filename: isProd ? 'bundle@[chunkhash].js' : 'bundle.js',
  },
  mode: ENV,
};
```

- 为不同环境创建各自的配置文件
  e.g., webpack.production.config.js ...

```
{
  ...
  "scripts": {
    "dev": " webpack-dev-server --config=webpack.development.config.js",
    "build": " webpack --config=webpack.production.config.js"
  },
}
```

可以把common提起到一个配置中
使用webpack-merge合并配置

# 开启production模式

webpack4 直接有mode配置项

## 环境变量

生产环境与本地环境添加不同的环境变量
Webpack中可以使用DefinePlugin进行设置

```
// webpack.config.js
const webpack = require('webpack');
module.exports = {
    entry: './app.js',
    output: {
        filename: 'bundle.js',
    },
    mode: 'production',
    plugins: [
        new webpack.DefinePlugin({
            ENV: JSON.stringify('production'),
        })
    ],
};

// app.js
document.write(ENV);
```

如果启用mode则不需要认为添加了

## source map

source map 指编译打包压缩后代码映射回源代码的过程

webpack打包后的代码可读性很差，调试代价很大，有了source map 可以使用浏览器调试工具

source map会跟随源码一步步被传递，直到最后生成的map文件。即打包文件后加上.map

生成mapping文件的同时，bundle也会追加注释来表示map文件的位置

```
// bundle.js
(function() {
  // bundle 的内容
})();
//# sourceMappingURL=bundle.js.map
```

不打开开发者工具，浏览器会无视map文件。

安全隐患，任何人都能通过开发者工具看到工程源码

### 配置

只需要webpack.config.js中加入devtool即可

```
module.exports = {
    // ...
    devtool: 'source-map',
};
```

对于css，scss，Less则需要添加额外配置项

source map 有好多形式

- cheap-source-map
- eval-source-map
  等等

### 安全

Webpack提供了 hidden-source-map 以及 nosource-source-map 提升source map的安全性

- hidden-source-map
  然后通过第三方服务 Sentry上传map
- nosources-source-map
  可以看到目录，但是看不到文件
  能在console控制台看到源码错误栈

## 资源压缩

代码压缩 uglify

- 移除多余空格、换行以及执行不到的代码
- 缩短变量名

压缩后代码可读性很差，提高了代码安全性

### 压缩js

- UglifyJS webpack3集成
- terser webpack4集成 支持es6+压缩 terser-webpack-plugin

### 压缩CSS

使用extract-text-webpack-plugin / mini-css-extract-plugin 将样式提取出来
用 optimize-css-assets-webpack-plugin 进行压缩
本质上使用的压缩器时 cssnano

## 缓存

重复利用浏览器已经获取过的资源

如何让用户不使用旧的缓存?
更新资源URL

### 资源hash

可以用chunkhash作为版本号，这样可以为每一个chunk单独计算一个hash，如果文件没变化则不会改变hash值

### 动态输出HTML

手动维护资源路径很困难
可以在打包的时候把资源名同步过去
html-webpack-plugin 可以做到这一点

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
module.exports = {
    // ...
    plugins: [
        new HtmlWebpackPlugin()
    ],
};
```

本质，生成一个新的index，把资源丢到index中，就不需要手动更新URL了

### chunk id 更稳定

Webpack3 模块id的变化会影响hash从而影响缓存
HashedModuleIdsPlugin 可以更改id生成方式
根据模块路径hash生成 id

Webpack 4修改了模块ID生成机制，不会有这个问题

## bundle体积监控和分析

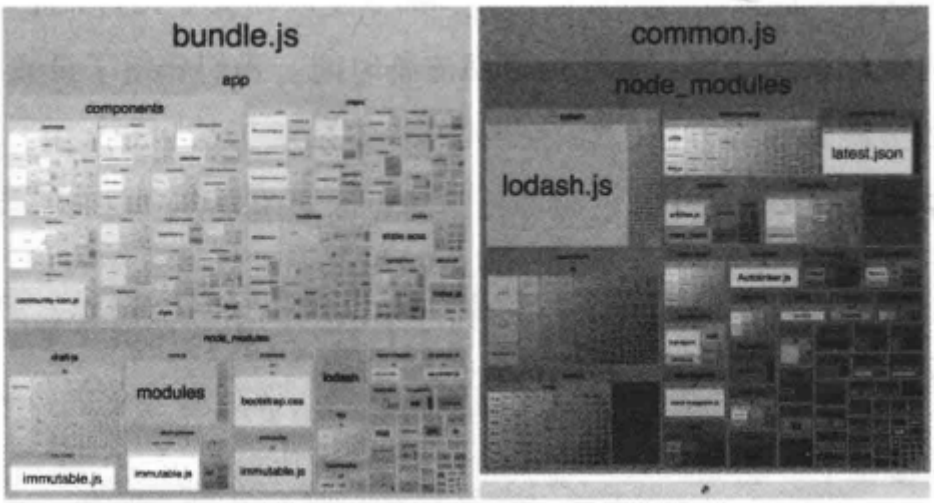
对打包输出的bundle体积进行监控，防止不必要的冗余模块被添加进来

vs code 中 Import Cost 可以监控引入模块的大小
可以计算压缩后以及gzip后占多大体积

webpack-bundle-analyzer 可以分析bundle的构成

```
const Analyzer = require('webpack-bundle-analyzer').BundleAnalyzerPlugin;
module.exports = {
    // ...
    plugins: [
        new Analyzer()
    ],
};
```

可以生成bundle模块组成的结构图，体积一目了然




bundlesize可以自动化对资源体积进行监控
package.json进行配置

# 总结

生产环境特殊配置
打包压缩
缓存
source map 以及安全性问题