webpack相关

1 打包体积 优化思路

- 提取第三方库或通过引用外部文件的方式引入第三方库
- 代码压缩插件`UglifyJsPlugin`
- 服务器启用gzip压缩
- 按需加载资源文件 `require.ensure`
- 优化`devtool`中的`source-map`
- 剥离`css`文件，单独打包
- 去除不必要插件，通常就是开发环境与生产环境用同一套配置文件导致

2 打包效率

- 开发环境采用增量构建，启用热更新
- 开发环境不做无意义的工作如提取`css`计算文件hash等
- 配置`devtool`
- 选择合适的`loader`
- 个别`loader`开启`cache`如`babel-loader`
- 第三方库采用引入方式
- 提取公共代码
- 优化构建时的搜索路径 指明需要构建目录及不需要构建目录
- 模块化引入需要的部分

3 Loader

编写一个loader

> `loader`就是一个`node`模块，它输出了一个函数。当某种资源需要用这个`loader`转换时，这个函数会被调用。并且，这个函数可以通过提供给它的`this`上下文访问`Loader API`。 `reverse-txt-loader`

```
// 定义
module.exports = function(src) {
  //src是原文件内容（abcde），下面对内容进行处理，这里是反转
  var result = src.split('').reverse().join('');
  //返回JavaScript源码，必须是String或者Buffer
  return `module.exports = '${result}'`;
}
//使用
```

```
{
    test: /\.txt$/,
    use: [
        {
            './path/reverse-txt-loader'
        }
    ]
},
```

4 说一下webpack的一些plugin，怎么使用webpack对项目进行优化

构建优化

- 减少编译体积 `ContextReplacementPugin`、`IgnorePlugin`、`babel-plugin-import`、`babel-plugin-transform-runtime`
- 并行编译 `happypack`、`thread-loader`、`uglifyjsWebpackPlugin`开启并行
- 缓存 `cache-loader`、`hard-source-webpack-plugin`、`uglifyjsWebpackPlugin`开启缓存、`babel-loader`开启缓存
- 预编译 `dllWebpackPlugin && DllReferencePlugin`、`auto-dll-webapck-plugin`

性能优化

- 减少编译体积 `Tree-shaking`、`Scope Hositing`
- `hash`缓存 `webpack-md5-plugin`
- 拆包 `splitChunksPlugin`、`import()`、`require.ensure`