# to-do tasks:

## gulp-flow:

0. html处理
    - 压缩
1. 图片处理
    - 压缩
    - 精灵图合并
    - 响应式处理：不同屏幕手机请求大小不同的尺寸
2. css处理
    - 压缩: 移除注释，空格
    - base64处理
    - 合并（备选）
3. js处理
    - 压缩
    - 合并（备选）

    **JS**文件的合并需:

    i. 页面依赖：这个页面的展示或者功能有必要加载这个文件吗？

    ii. 依赖程度：页面对它的依赖程度多高？需要一开始就发起请求吗？

    iii. 考虑页面的依赖，考虑加载，既不会影响功能，又不影响用户体验。

    考虑到当前**fw.all.js**过重，压缩比不高，可以考虑拆分后合并。进一步减小它的加载时间（2.5s）：

    - hightchart 和echart图片冲突，考虑只使用一个。
4. 项目增量处理
    - 只对变动的文件处理，而不是全局处理
    - 全局监听文件，配合SVN更新代码就监听变化，进而触发增量处理
5. 开发调试

sourcemap建立源文件与构建文件的映射关系，使得调试方便，若为开发模式，就开启sourcemap生成，如为发布模式，可关闭sourcemap生成，针对项目而言，我们需要在发布前就行增量处理，开发阶段不需要关心构建过程，所以，配置可以设置为发布模式，即取消sourcemap生成。

要尝试使用带sourcemap的构建，可以将sourcemap文件写到特定文件夹中，无需使用默认方式

6. 开发与发布的分离

开发的配置与发布的工作流不同：

- 开发(config.dev)：
    - sass实时编译
    - 浏览器同步刷新
    - 响应式图片自动化切割
    - 精灵图合并与css自动生成
    - base64编译
    - sourcemap辅助调试
- 发布(config.build)
    - 代码压缩
    - 静态资源压缩
    - 监听增量任务

## lazysizes

实现懒加载
举个栗子：

首页banner图，就有两个图的请求，如果，业务上不容易实现直接，可以考虑，首先用小图填充(1 * 1 px)的图片，这样也能减少资源的加载。

## 预测加载

预计用户行为加载

## 首屏输出、白屏

首屏依赖资源的加载优化

对首页加载的fw.all.js进行拆分，合并首页展现必须的文件

- progress.js: 首页进度图

- modernizr: 特性组合探测库
- isScroll: 滑动特效等
- Hightcharts/echarts: 数据可视化库
- touchjs.min：手势库
- H5lock: 锁屏库
- twemoji：表情库
- swiper： slide库

# 首页到发现页跳转卡顿

# 构建测试

1. 文件大小，流量控制
2. 压缩是否会导致功能的缺失

# 存在问题

## 动画压缩问题

1.压缩css文件时，将index.css中的动画rotateIn定义移除了
2.animation.css压缩，变量混淆后，loading图的动画名rotateIn(这个库刚好也是定义的这个动画名)替代为混淆后的动画名C，直接导致动画旋转变为平移

~~解决办法：对于动画库（引入的插件库）直接下载官方min版本，在构建过程中排除掉，不处理。~~

解决办法：插件管理，options对象传入

cssnano的优化选项