1. 前端性能优化原则：1.少加载文件，2.少执行代码
2. 利用浏览器缓存，强缓存，协议（对比）缓存
3. 启用cdn，把文件放到cdn服务器中，文件分发到各地，减少⽤户和服务器的距离
4. 具体的优化项目

一、减少请求资源大小或者次数

1、尽量和并和压缩css和js文件。（将css文件和并为一个。将js合并为一个）

　　原因：主要是为了减少http请求次数以及减少请求资源的大小

　　打包工具：

　　webpack

　　gulp

　　grunt

.　　....

2、尽量所使用的字体图标或者SVG图标来代替传统png图

　　因为字体图标或者SVG是矢量图，代码编写出来的，方大不会变形，而且渲染速度快

3、采用图片的懒加载（延迟加载）

　　目的为了，减少页面第一次加载过程中http的请求次数

　　具体步骤：

　　　　1、页面开始加载时不去发送http请求，而是放置一张占位图

　　　　2、当页面加载完时，并且图片在可视区域再去请求加载图片信息

4、能用css做的效果，不要用js做，能用原生js做的，不要轻易去使用第三方插件。

　　避免引入第三方大量的库。而自己却只是用里面的一个小功能

5、使用雪碧图或者是说图片精灵

　　把所有相对较小的资源图片，绘制在一张大图上，只需要将大图下载下来，然后利用

　　图片定位来讲小图展现在页面中（background-position:百分比，数值）

6、减少对cookie的使用（最主要的就是减少本地cookie存储内容的大小），因为客户端操作cookie的时候，这些信息总是在客户端和服务端传递。如果上设置不当，每次发送

一个请求将会携带cookie

8、前端与后端协商，合理使用keep-alive

9、前端与服务器协商，使用响应资源的压缩

10、避免使用iframe

　　不仅不好管控样式，而且相当于在本页面又嵌套其他页面，消耗性能会更大。因为还回去加载这个嵌套页面的资源

11、在基于ajax的get请求进行数据交互的时候，根据需求可以让其产生缓存（注意:这个

缓存不是我们常看到的304状态码，去浏览器本地取数据），这样在下一次从相同地址获取是数据

时，取得就是上一次缓存的数据。（注意：很少使用，一般都会清空。根据需求来做）

二、代码优化相关

1、在js中尽量减少闭包的使用

　　原因：使用闭包后，闭包所在的上下文不会被释放

2、减少对DOM操作，主要是减少DOM的重绘与回流（重排）

　　关于重排（回流）的分离读写：如果需要设置多个样式，把设置样式全放在一起设置，不要一条一条的设置。使用文档碎片或者字符串拼接做数据绑定（DOM的动态创建）

3、在js中避免嵌套循环和"死循环"(一旦遇到死循环，浏览器就会直接卡掉)

4、把css放在body上，把js放在body下面

　　让其先加载css（注意：这里关于优化没有多大关系）

5、减少css表达式的使用

6、css选择器解析规则所示从右往左解析的。减少元素标签作为对后一个选择对象

7、尽量将一个动画元素单独设置为一个图层（避免重绘或者回流的大小）

　　注意：图层不要过多设置，否则不但效果没有达到反而更差了

8、在js封装过程中，尽量做到低耦合高内聚。减少页面的冗余代码

9、css中设置定位后，最好使用z-index改变盒子的层级，让盒子不在相同的平面上

10、css导入的时候尽量减少@import导入式，因为@import是同步操作，只有把对应的样式导入后，才会继续向下加兹安，而link是异步的操作

11、使用window.requestAnimationFrame(js的帧动画)代替传统的定时器动画

　　如果想使用每隔一段时间执行动画，应该避免使用setInterval，尽量使用setTimeout

　　代替setInterval定时器。因为setInterval定时器存在弊端：可能造成两个动画间隔时间

　　缩短

12、尽量减少使用递归。避免死递归

　　解决：建议使用尾递归

13、基于script标签下载js文件时，可以使用defer或者async来异步加载

14、在事件绑定中，尽可能使用事件委托，减少循环给DOM元素绑定事件处理函数。

15、在js封装过程中，尽量做到低耦合高内聚。减少页面的冗余代码