一、浏览器渲染引擎

主要模块

- 一个渲染引擎主要包括：HTML解析器，CSS解析器，javascript引擎，布局layout模块，绘图模块
  - HTML解析器：解释HTML文档的解析器，主要作用是将HTML文本解释成DOM树。
  - CSS解析器：它的作用是为DOM中的各个元素对象计算出样式信息，为布局提供基础设施
  - Javascript引擎：使用Javascript代码可以修改网页的内容，也能修改css的信息，javascript引擎能够解释javascript代码，并通过DOM接口和CSS树接口来修改网页内容和样式信息，从而改变渲染的结果。
  - 布局（layout）：在DOM创建之后，Webkit需要将其中的元素对象同样式信息结合起来，计算他们的大小位置等布局信息，形成一个能表达这所有信息的内部表示模型
  - 绘图模块（paint）：使用图形库将布局计算后的各个网页的节点绘制成图像结果

> 备注：文档对象模型（Document Object Model，简称DOM）

大致的渲染过程

参考如下图片：

- 浏览器渲染页面的整个过程：浏览器会从上到下解析文档。
  1. 遇见 HTML 标记，调用HTML解析器解析为对应的 token （一个token就是一个标签文本的序列化）并构建 DOM 树（就是一块内存，保存着tokens，建立它们之间的关系）。
  2. 遇见 style/link 标记调用相应解析器处理CSS标记，并构建出CSS样式树。
  3. 遇见 script 标记 调用javascript引擎 处理script标记、绑定事件、修改DOM树/CSS树等
  4. 将 DOM树 与 CSS树 合并成一个渲染树。
  5. 根据渲染树来渲染，以计算每个节点的几何信息（这一过程需要依赖GPU）。
  6. 最终将各个节点绘制到屏幕上。

> 以上这些模块依赖很多其他的基础模块，包括要使用到网络 存储 2D/3D图像 音频视频解码器 和图片解码器。
> 所以渲染引擎中还会包括如何使用这些依赖模块的部分。

二、阻塞渲染

1.关于css阻塞：

声明：只有link引入的外部css才能够产生阻塞。

1. style标签中的样式：

```
(1). 由html解析器进行解析 (html解析器是异步解析的)；
(2). 页面style标签写的内部样式是异步解析的，容易产生闪屏现象；
     - 异步解析时，结构解析完但样式还没有解析完
     - 例如：结构只有一个div，但样式对这个div修改了6万次颜色，结构很快就解析完了，用户会看到盒子一直再闪。
(3). 浏览器加载资源是异步的；
(4). 不阻塞浏览器渲染；
(5). 不阻塞DOM解析；
```

2. link引入的外部css样式（推荐使用的方式）：

```
(1). 由CSS解析器进行解析 (CSS解析器是同步解析的)；
(2). 阻塞浏览器渲染(即DOM解析完要等待CSS解析完才可以在页面显示，可以利用这种阻塞避免“闪屏现象”)；
(3). 阻塞其后面的js语句的执行；(CSS能设置样式，JS也能，避免对同一个元素设置样式冲突)
(4). 不阻塞DOM的解析(绝大多数浏览器的工作方式)
```

3. 优化核心理念：尽可能快的提高外部css加载速度

```
(1). 使用CDN节点进行外部资源加速。
(2). 对css进行压缩(利用打包工具，比如webpack,gulp等)。
(3). 减少http请求数，将多个css文件合并。
(4). 优化样式表的代码
```

2.关于js阻塞：

```
1. 阻塞后续DOM解析：
   原因：浏览器不知道后续脚本的内容，如果先去解析了下面的DOM，而随后的js删除了后面所有的DOM，
         那么浏览器就做了无用功，浏览器无法预估脚本里面具体做了什么操作，例如像document.write
         这种操作，索性全部停住，等脚本执行完了，浏览器再继续向下解析DOM。
2. 阻塞页面渲染：
   原因：js中也可以给DOM设置样式，浏览器等该脚本执行完毕，渲染出一个最终结果，避免做无用功。
3. 阻塞后续js的执行：
   原因：维护依赖关系，例如：必须先引入jQuery再引入bootstrap
```

3.备注

```
【备注1】：css的解析和js的执行是互斥的（互相排斥），css解析的时候js停止执行，js执行的时候css停止解析。

【备注2】：无论css阻塞，还是js阻塞，都不会阻塞浏览器加载外部资源（图片、视频、样式、脚本等）
          原因：浏览器始终处于一种：“先把请求发出去”的工作模式，只要是涉及到网络请求的内容，
                无论是：图片、样式、脚本，都会先发送请求去获取资源，至于资源到本地之后什么时候用，
                由浏览器自己协调。这种做法效率很高。

【备注3】：WebKit 和 Firefox 都进行了【预解析】这项优化。在执行js脚本时，浏览器的其他线程会预解析文档的其余部分，
           找出并加载需要通过网络加载的其他资源。通过这种方式，资源可以在并行连接上加载，从而提高总体速度。请注意，
           预解析器不会修改 DOM 树
```

> 在上述的过程中，网页在加载和渲染过程中会触发“DOMContentLoaded”和“onload”事件
> 分别是在DOM树构建（解析）完成之后，以及DOM树构建完并且网页所依赖的资源都加载完之后

- 上面介绍的是一个完整的渲染过程，但现代网页很多都是动态的，这意味着在渲染完成之后，由于网页的动画或者用户的交互，
  浏览器其实一直在不停地重复执行渲染过程。（重绘重排），以上的数字表示的是基本顺序，这不

是严格一致的，
这个过程可能重复也可能交叉