2. Node：根据标记创建节点
3. CSSOM：节点创建CSSOM树

21. 根据DOM树和CSSOM树构建渲染树：
    1. 从DOM树的根节点遍历所有可见节点，不可见节点包括：1) `script`，`meta` 这样本身不可见的标签。2)被css隐藏的节点， 如 `display: none`
    2. 对每一个可见节点，找到恰当的CSSOM规则并应用
    3. 发布可视节点的内容和计算样式
22. js解析如下：
    1. 浏览器创建Document对象并解析HTML，将解析到的元素和文本节点添加到文档中，此时document.readystate为loading
    2. HTML解析器遇到没有async和defer的script时，将他们添加到文档中，然后执行行内或外部脚本。这些脚本会同步执行， 并且在脚本下载和执行时解析器会暂停。这样就可以用document.write()把文本插入到输入流中。同步脚本经常简单定义函数和注册事件处理程序，他们可以遍历和操作script和他们之前的文档内容
    3. 当解析器遇到设置了async属性的script时， 开始下载脚本并继续解析文档。脚本会在它下载完成后尽快执行，但是解析器不会停下来等它下载。异步脚本禁止使用document.write()， 它们可以访问自己script和之前的文档元素
    4. 当文档完成解析，document.readState变成interactive
    5. 所有defer脚本会按照在文档出现的顺序执行，延迟脚本能访问完整文档树， 禁止使用document.write()
    6. 浏览器在Document对象上触发DOMContentLoaded事件
    7. 此时文档完全解析完成， 浏览器可能还在等待如图片等内容加载， 等这些内容完成载入并且所有异步脚本完成载入和执行，document.readState变为complete，window触发load事件
23. 显示页面（HTML解析过程中会逐步显示页面)

详细简版

1. 从浏览器接收 `url` 到开启网络请求线程（这一部分可以展开浏览器的机制以及进程与线程之间的关系)

2. 开启网络线程到发出一个完整的 `HTTP` 请求（这一部分涉及到dns查询， `TCP/IP` 请求，五层因特网协议栈等知识)

3. 从服务器接收到请求到对应后台接收到请求（这一部分可能涉及到负载均衡， 安全拦截以及后台内部的处理等等)

4. 后台和前台的 `HTTP` 交互（这一部分包括 `HTTP` 头部、响应码、报文结构、 `cookie` 等知识， 可以提下静态资源的 `cookie` 优化， 以及编码解码， 如 `gzip` 压缩等)