1. 输入一个URL到渲染页面的过程
2. 输入URL，浏览器查找当前URL是否存在缓存，并比较缓存是否过期
3. 强缓存：请求返回的资源会进行缓存，下次请求直接使用缓存，一般会返回一个失效时间（Max-age）
4. 协商缓存：请求返回的资源会进行缓存，下次请求在使用缓存之前需要与服务器进行交互对比缓存的资源是否有更新，如果有更新返回200新的资源，如果没有更新返回304 Not-Modified,直接使用缓存资源。

2.DNS解析域名成IP地址

3.根据IP地址向对应的服务器发送网络请求

(建立TCP连接三次握手）

4.服务器处理请求，返回对应的文件

（通常是返回index.html, 解析过程需要加载对应的css，js文件再去请求返回）

5.浏览器解析文件并渲染页面

渲染过程

（1）HTML Parse模块解析HTML文件，生成DOM Tree

（2）解析过程如果碰到script标签，会停止解析HTML，执行完JS再去解析

1. 即按HTML顺序来执行，但是当JS文件过大，执行时间过长会阻塞页面，即等待时间过长
2. 解决方式一： 将scrpit脚本放在页面底部，但是这种方式也存在问题，浏览器解析完HTML才会去加载对应的脚本，对于长的 HTML 文档来说，这样可能会造成明显的延迟
3. 解决方式二：加载异步脚本，这里script标签有两个特性async，defer
4. defer特性仅适用于外部脚本，即带有src的脚本
5. defer特性会告诉浏览器不要执行脚本，继续解析HTML，并且继续下载JS脚本，等到DOM解析完毕之后，并且在DOMContentLoaded事件触发之前执行脚本。（DOMContentLoaded事件--浏览器解析完HTML,构建完DOM Tree，但是像img这样的外部资源可能尚未加载完毕）
6. async特性也是异步加载脚本。
7. defer，async的区别：

defer特性可以保证脚本的先后执行顺序，即在文档靠前的先执行。

async特性不可以保证顺序，先加载完的先执行,加载完毕就执行，不会等待事件触发。

1. 当我们在JS脚本中添加动态脚本，即动态添加script标签时，也是异步加载的，默认添加async特性，可以添加script.async = false让其与defer一样。

（3）CSS Parse模块解析CSS，生成CSSOM（对应的Style Rules）

（4）将DOM Tree与Style Rules附加在一起，生成Render Tree（渲染树）

（5）根据Render Tree进行布局

（6）渲染对应的页面进行展示

6.关闭TCP连接（四次挥手）

1. 浏览器重排和重绘
2. 重排（回流）：当页面元素发生变化（尺寸大小，定位，显示隐藏等），影响到页面布局时，此时Render Tree需要重新构建，这个过程就是重排
3. 重绘：是指元素外观发生变化（颜色等），并不影响页面布局时，Render Tree不需要重新构建，只需要根据对应的Style Rules重新绘制元素即可。
4. 重绘和重排的关系: 重排一定会引发重绘，即重新构建完Render Tree之后一定会重新绘制元素。
5. 引发重排的方式：
6. 浏览器大小改变（resize）
7. Animation，position，margin等会改变元素尺寸大小的属性
8. Display：none 隐藏元素
9. 读取某些元素属性：（offsetLeft/Top/Height/Width,　clientTop/Left/Width/Height, scrollTop/Left/Width/Height,　width/height,　getComputedStyle())
10. 页面初次渲染
11. 引发重绘的方式
12. Opacity, visibility,color等
13. Display:none 不会占据空间 会引起重排和重绘
14. Visibility:hidden 会占据空间 会引起重绘 不可以触发单击事件
15. Opacity:0 会占据空间 可以触发单击事件
16. V8引擎
17. 浏览器内核包括两个部分，渲染层和JS引擎
18. V8引擎工作原理

（JS代码经过字符编码处理被Scanner以流的方式被Scanner进行读取）

1. JS代码经过词法分析生成tokens（Scanner）

Tokens: [{ type: ‘keyword’, value: ‘const’}…]

1. 经过Parse模块语法分析然后生成抽象语法树（ast）

（不执行的代码会进行预解析Lazy Parsing，执行时才真正的进行解析生成ast）

1. Ast由ignition模块生成字节码（bytecode）

生成字节码主要是为了适配多平台（不同平台不同架构的cpu机器指令不一样）

1. 把字节码编译成汇编代码，再把汇编代码编译成机器指令，交由CPU去执行
2. 这个过程V8引擎做了优化，ignition模块会收集高频率执行的JS代码信息交由TurboFan这个模块来直接编译成优化过的机器码去执行，当出现不同操作时会将对应的机器码重新转换成字节码，然后重新去执行。

举例有一个sum函数（调用了很多次此时就会被标记为热点函数被TurboFan转化成优化的机器码），接收两个number类型的参数，返回两个数相加的结果，但是当传入的类型变成string时这个时候sum函数需要返回字符串拼接的结果，此时就需要将优化后的机器码转换成字节码重新执行。（TypeScript代码会进行类型检测，所以编译出来的JavaScript代码运行效率会比原来的JS代码高）

1. Vue template编译原理
2. 将template编译成抽象语法树（ast）
3. 将ast编译成对应的render函数（render函数的返回值为vnode）
4. Babel编译原理
5. Babel执行阶段

源代码 – 解析 – 转换 – 代码生成 – 目标源代码

1. Babel编译原理

源代码 – 词法分析 – 生成tokens数组 – 语法分析 – 生成ast – 遍历ast – 访问ast的节点 – 应用插件 – 生成新的ast – 生成目标源代码