从敲入 URL 到浏览器渲染完成

2.1 输入地址

浏览器引入了 DNS 预取技术。它是利用现有的 DNS 机制，提前解析网页中可能的网络连接。

当我们开始在浏览器中输入网址的时候，浏览器其实就已经在智能的匹配可能得 url 了。它会从历史记录，书签等地方，找到已经输入的字符串可能对应的 url ，找到同输入的地址很匹配的项，然后给出智能提示，让你可以补全 url 地址。用户还没有按下 enter 键， 浏览器已经开始使用 DNS 预取技术解析该域名了。

对于 chrome 的浏览器，如果有该域名相关的缓存，它会直接从缓存中把网页展示出来，就是说，你还没有按下 enter，页面就出来了。如果没有缓存，就还是会重新请求资源。

2.2 查询 DNS 查找对应的请求 IP 地址

假设输入 www.baidu.com，大概过程：

浏览器搜索自己的 DNS 缓存。

在浏览器缓存中没找到，就在操作系统缓存中查找，这一步中也会查找本机的 hosts 看看有没有对应的域名映射。

在系统中也没有的话，就到你的路由器来查找，因为路由器一般也会有自己的 DNS 缓存。

若没有，则操作系统将域名发送至 本地域名服务器——递归查询方式，本地域名服务器 查询自己的 DNS 缓存，查找成功则返回结果，否则，采用迭代查询方式。本地域名服务器一般都是你的网络接入服务器商提供，比如中国电信，中国移动。

本地域名服务器 将得到的 IP 地址返回给操作系统，同时自己也将 IP 地址缓存起来。

操作系统将 IP 地址返回给浏览器，同时自己也将 IP 地址缓存起来，以备下次别的用户查询时，可以直接返回结果，加快网络访问。

至此，浏览器已经得到了域名对应的 IP 地址。

2.3 建立 TCP 连接

TCP 是一种面向有连接的传输层协议。 它可以保证两端（发送端和接收端）通信主机之间的通信可达。 它能够处理在传输过程中丢包、传输顺

序乱掉等异常情况；此外它还能有效利用宽带，缓解网络拥堵。
三次握手的步骤：（抽象派）

客户端：hello，你是server么?
服务端：hello，我是server，你是client么
客户端：yes，我是client
复制代码

在 TCP 连接建立完成之后就可以发送 HTTP 请求了。
然后，待到断开连接时，需要进行四次挥手（因为是全双工的，所以需要四次挥手）
四次挥手的步骤：（抽象派）

主动方：我已经关闭了向你那边的主动通道了，只能被动接收了
被动方：收到通道关闭的信息
被动方：那我也告诉你，我这边向你的主动通道也关闭了
主动方：最后收到数据，之后双方无法通信
复制代码

2.4 服务器收到请求并响应 HTTP 请求
在接收和解释请求消息后，服务器返回一个HTTP响应消息。
HTTP 响应由三个部分组成，分别是：**状态行、消息报头、响应正文。**
状态代码：由三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：

1xx：指示信息--表示请求已接收，继续处理
2xx：成功--表示请求已被成功接收、理解、接受
3xx：重定向--要完成请求必须进行更进一步的操作
4xx：客户端错误--请求有语法错误或请求无法实现
5xx：服务器端错误--服务器未能实现合法的请求

常见状态代码、状态描述、说明：

200 OK： 客户端请求成功
400 Bad Request： 客户端请求有语法错误，不能被服务器所理解
401 Unauthorized： 请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用
403 Forbidden： 服务器收到请求，但是拒绝提供服务
404 Not Found： 请求资源不存在，eg：输入了错误的URL

500 Internal Server Error： 服务器发生不可预期的错误
503 Server Unavailable： 服务器当前不能处理客户端的请求，一段时间后可能恢复正常

HTTP消息报头包括：普通报头、请求报头、响应报头、实体报头。具体不作介绍。
响应正文：就是服务器返回的资源的内容

2.5 浏览器接收服务器响应结果并处理

在浏览器没有完整接受全部HTML文档时，它就已经开始显示这个页面了，不同浏览器可能解析的过程不太一样，这里我们只介绍 WebKit 的渲染过程。
渲染步骤大致可以分为以下几步：

1. 解析HTML，构建 DOM 树
2. 解析 CSS ，生成 CSS 规则树
3. 合并 DOM 树和 CSS 规则，生成 render 树
4. 布局 render 树（ Layout / reflow ），负责各元素尺寸、位置的计算
5. 绘制 render 树（ paint ），绘制页面像素信息
6. 浏览器会将各层的信息发送给 GPU，GPU 会将各层合成（ composite ），显示在屏幕上

其中每个解释的过程中，WebKit 都提供了很多相关的类来一步一步地解释对应的内部模块，这里面不做详细描述。
下面根据上面的大致过程来一步步细解。

2.5.1 构造 DOM 树

浏览器在解析html文件时, 是WebKit 中的 HTML 解释器的将网络或者本地磁盘获取的 HTML 网页和资源从字节流解释成 DOM 树结构。具体过程如下：

在 WebKit 中这一过程如下：首先是字节流，经过解码之后是字符流，然后通过词法分析器会被解释成词语（Tokens），之后经过语法分析器构建成节点，最后这些节点被组建成一棵 DOM 树。
浏览器在解析html文件过程中，会 ”自上而下“ 加载，并在加载过程中进行解析渲染。在解析过程中，如果遇到请求外部资源时，如图片、外链的CSS、iconfont等，请求过程是异步的，并不会影响html文档进行加载，且统一交由 Browser 进程来处理，这使得资源在不同网页间的共享变得很容易。

HTML 的解释、布局和渲染等工作基本上就是工作在渲染线程完成的（这不是绝对的）。因为 DOM 树只能在渲染线程上创建和访问，这也就是说构建 DOM 树的过程只能在渲染线程中进行，但是，从字符到词语这个阶段可以交给另外的单独的线程来做。
而且因为有 DNS 预取技术，当用户正在浏览当前网页的时候，Chromium 提取网页中的超链接，将域名抽取出来，利用比较少的 CPU 和网络带宽来解析这些域名或者 IP 地址，这样一来，用户根本感觉不到这一过程。当用户单击这些链接的时候，可以节省不少时间，特别在域名解析比较慢的时候，效果特别明显。
解析过程中，**浏览器首先会解析 HTML 文件构建 DOM 树，然后解析 CSS 文件构建 Render树，等到 Render 树构建完成后，浏览器开始布局 Render 树并将其绘制到屏幕上。**

2.5.2 解释 CSS

CSS 解释过程是指从 CSS 字符串 经过 CSS 解释器 处理后变成渲染引擎内部规则的表示过程。
生成样式规则之后，会进行样式规则匹配，WebKit 会为其中的一些节点（只限于可视节点）选择合适的样式信息，规则的匹配则是由 ElementRuleCollector 类来计算并获得，它根据元素的属性等，并从 DocumentRuleSets 类中获取规则集合，依次按照 ID、类别、标签等选择器信息逐次匹配获得元素的样式。
最后，WebKit 对这些规则进行排序。对于该元素需要的样式属性，WebKit 选择从高优先级规则中选取，并将样式属性值返回。
从整个网页的加载和渲染过程来看，CSS 解释和规则匹配处于 DOM 树建立之后，RenderObject 树建立之前，CSS 解释器解释后的结果会保存起来，然后 RenderObject 树基于该结果来进行规范匹配和布局计算。当网页有用户交互或者动画等动作的时候，通过 CSSDOM 等技术，JavaScript 代码同样可以非常方便地修改 CSS 代码，WebKit 此时需要重新解释样式并重复以上这一过程。

2.5.3 渲染过程遇到 JavaScript

当文档加载过程中遇到 js 文件，html 文档会挂起渲染（加载解析渲染同步）的线程，不仅要等待文档中 js 文件加载完毕，还要等待解析执行完毕，才可以恢复 html 文档的渲染线程。因为 JS 有可能会修改 DOM，最为经典的 document.write，这意味着，在 JS 执行完成前，后续所有资源的下载可能是没有必要的，这是 js 阻塞后续资源下载的根本原因。所以

我们平时的代码中，js 是放在 html 文档末尾的。
而且当遇到执行 JavaScript 代码的时候，WebKit 先暂停当前 JavaScript 代码的执行，使用预先扫描器 HTMLPreloadScanner 类来扫描后面的词语。如果 WebKit 发现它们需要使用其他资源，那么使用预资源加载器 HTMLPreloadScanner 类来发送请求，在这之后，才执行 JavaScript 代码。预先扫描器本身并不创建节点对象，也不会构建 DOM 树，所以速度比较快。
当 DOM 树构建完之后，WebKit 触发 “DOMContentLoaded” 事件，注册在该事件上的 JavaScript 函数会被调用。当所在资源都被加载完之后，WebKit 触发 “onload” 事件。
WebKit 将 DOM 树创建过程中需要执行的 JavaScript 代码交由 HTMLScriptRunner 类来负责。工作方式很简单，就是利用 JavaScript 引擎来执行 Node 节点中包含的代码。
JS 的解析是由浏览器中的 JavaScript 引擎完成的。JS是单线程运行，也就是说，在同一个时间内只能做一件事，所有的任务都需要排队，前一个任务结束，后一个任务才能开始。但是又存在某些任务比较耗时，如 IO 读写等，所以需要一种机制可以先执行排在后面的任务，这就是：同步任务(synchronous)和异步任务(asynchronous)。
JS 的执行机制就可以看做是一个主线程加上一个任务队列(task queue)。同步任务就是放在主线程上执行的任务，异步任务是放在任务队列中的任务。所有的同步任务在主线程上执行，形成一个执行栈; 异步任务有了运行结果就会在任务队列中放置一个事件；脚本运行时先依次运行执行栈，然后会从任务队列里提取事件，运行任务队列中的任务，这个过程是不断重复的，所以又叫做事件循环(Event loop)。
参考小汪之前写的文章：浏览器之 javaScript 引擎

2.5.4 渲染合成 Render 树
HTML 经过 WebKit 解释之后，生成 DOM 树。在 DOM 树构建完成之后，WebKit 会为 DOM 树节点构建 RenderObject 树，再通过 RenderObject 树构建出 RenderLayer 树。
RenderObject 树是基于 DOM 树建立起来的一棵新树，是为了布局计算和渲染等机制而构建的一种新的内部表示。**RenderObject 树节点和 DOM 节点不是一一对应关系**，因为有可视节点（常用的 div img 标签等）与不可视节点（如 head、meta 标签），不可视节点是不会构成 RenderObject 树的。

网页是有层次结构的，可以分层的，一是为了方便设置网页的层次，二是为了 WebKit 处理上的便利，为了简化渲染的逻辑。
而且 **RenderLayer 节点和 RenderObject 节点不是一一对应关系，而是一对多的关系。**

2.5.5 布局

当 WebKit 创建 RenderObject 对象之后，每个对象是不知道自己的位置、大小等信息的，WebKit 根据框模型来计算它们的位置，大小等信息的过程称为布局计算。
布局计算是一个递归的过程，因为一个节点的大小通常需要先计算它的子女节点的位置，大小等信息。
当用户 网页的动画、翻滚网页、JavaScript 代码通过 CSSDOM 等操作时还会有重新布局。
参考小汪之前写的文章：浏览器内核之 CSS 解释器和样式布局

2.5.6 绘图

在 WebKit 中，绘图操作就是绘图上下文，所有绘图的操作都是在该上下文中来进行的。
绘图上下文可以分成两种类型：
一是 2D 图形上下文（GraphicsContext），用来绘制 2D 图形的的上下文；
二是 3D 绘图上下文，是用来绘制 3D 图形的上下文。
2D 绘图上下文具体的作用：提供基本绘图单元的绘制接口以及设置绘图的样式。绘图接口包括画点，画线、画图片、画多边形、画文字等，绘图样式包括颜色、线宽、字号大小、渐变等。
关于 3D 绘图上下文，它的主要用处是支持 CSS3D、WebGL 等。
网页的渲染方式，有三种方式，一是软件渲染，二是硬件加速渲染，三可以说是混合模式。

如果绘图操作使用 CPU 来完成，称之为软件绘图。
如果绘图操作由 GPU 来完成，称之为 GPU 硬件加速绘图。
理想情况下，每个层都有个绘制的存储区域，这个存储区域用来保存绘图的结果。最后，需要将这些层的内容合并到同一个图像之中，可以称之为合成（Compositing），使用了合成技术的渲染称之为合成化渲染。
所以，在完成构建 DOM 树之后，WebKit 会调用绘图操作、软件渲染或者硬件加速渲染或者两者都有，将模型绘制出来，呈现在屏幕上。 至

此，浏览器渲染完成。