How browsers work

http://taligarsiel.com/Projects/howbrowserswork1.htm

1.介绍

- 我们将讨论的浏览器

  目前使用的主要浏览器有五种——Internet Explorer、Firefox、Safari、Chrome 和 Opera。
  我将举一些开源浏览器的例子——Firefox、Chrome 和 Safari，它们是部分开源的。

- 浏览器的主要功能

  浏览器的主要功能是从服务器请求您选择的 Web 资源，并在浏览器窗口上显示呈现它。资源格式通常是 HTML，但也有 PDF、图像等。资源的位置由用户使用 URI（统一资源标识符）指定。

  常见的用户界面元素包括：

  > 用于插入 URI 的地址栏
  >
  > 后退和前进按钮
  >
  > 书签选项
  >
  > 刷新和停止按钮，用于刷新和停止当前文档的加载
  >
  > 主页按钮，可让您进入主页

  HTML5 规范没有定义浏览器必须具有的 UI 元素，但列出了一些常见元素。其中包括地址栏、状态栏和工具栏

- 浏览器的高层结构

  浏览器的主要组件是：

  1. 用户界面（The user interface） - 这包括地址栏、后退/前进按钮、书签菜单等。除了您看到请求页面的主窗口外，浏览器的每个部分都会显示。
  2. 浏览器引擎（The browser engine）——查询和操作渲染引擎的接口。
  3. 渲染引擎（The rendering engine）——负责显示请求的内容。例如，如果请求的内容是 HTML，它负责解析 HTML 和 CSS，并将解析后的内容显示在屏幕上。
  4. 网络（Networking） - 用于网络调用，如 HTTP 请求。它具有独立于平台的接口和每个平台的底层实现。
  5. UI 后端（UI backend） - 用于绘制基本小部件，如组合框和窗口。它公开了一个非平台特定的通用接口。在它下面使用操作系统用户界面方法。
  6. JavaScript 解释器（JavaScript interpreter）。用于解析和执行 JavaScript 代码。
  7. 数据存储（Data storage）。这是一个持久层。浏览器需要在硬盘上保存各种数据，例如cookies。新的 HTML 规范 (HTML5) 定义了“网络数据库”，它是浏览器中的一个完整（虽然很轻）的数据库。

  与大多数浏览器不同，Chrome 拥有多个渲染引擎实例 - 每个选项卡一个。每个选项卡都是一个单独的过程。

- 组件之间的通信

2. 渲染引擎

- 渲染引擎

  Firefox 使用 Gecko——一种“自制”的 Mozilla 渲染引擎。Safari 和 Chrome 都使用 Webkit。

- 主要流程

  1. Parsing HTML to construct the DOM tree:解析 HTML 文档并将标签转换为名为“内容树”的树中的 DOM节点
  2. Render tree construction:HTML 中的样式信息和可视化指令将用于创建另一棵树——渲染树
  3. Layout of the render tree:在构建渲染树之后，它会经历一个布局过程
  4. Painting the render tree:绘制- 将遍历渲染树，并使用 UI 后端层绘制每个节点

  这是一个渐进的过程。为了更好的用户体验，渲染引擎会尽量在屏幕上尽快显示内容。在开始构建和布局渲染树之前，它不会等到所有 HTML 都被解析。部分内容将被解析和显示，而该过程将继续处理来自网络的其余内容。

- 主要流程示例

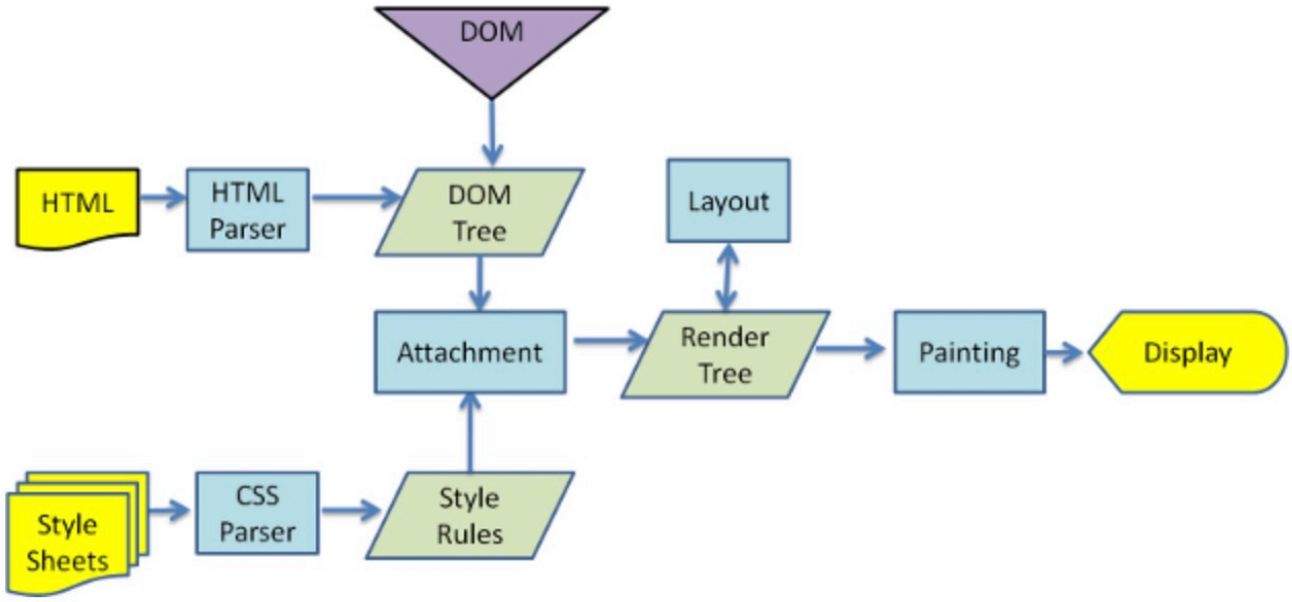

图 3：Webkit 主要流程

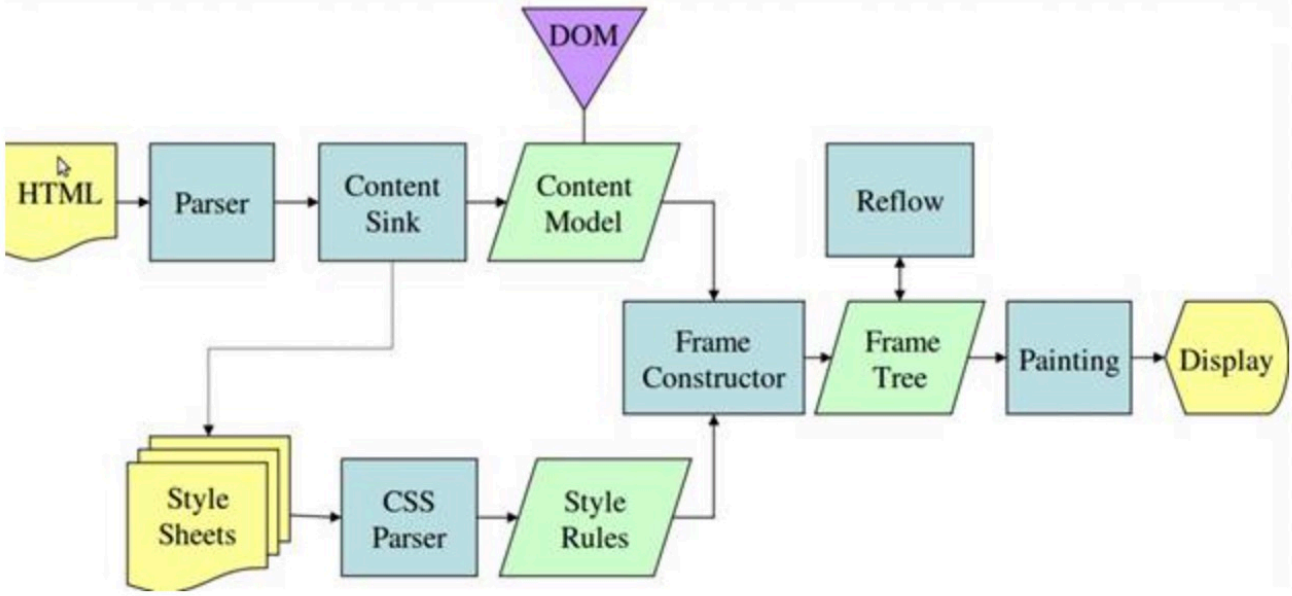

图 4：Mozilla 的 Gecko 渲染引擎主流程（3.6）

- 解析和DOM树构建
  - 解析-一般

    解析文档意味着将其翻译成有意义的结构——代码可以理解和使用的结构。解析的结果通常是代表文档结构的节点树。它被称为解析树或语法树。

    解析可以分为两个子过程：

    > 词法分析：将输入分解为标记的过程。标记是语言词汇——有效构建块的集合。
    >
    > 句法分析：语言语法规则的应用。

    解析器通常将工作分配给两个组件：

    > 词法分析器：负责将输入分解为有效标记
    >
    > 解析器：负责通过根据语言语法规则分析文档结构来构建解析树

解析过程是迭代的。

很多时候，解析树不是最终产品。解析常用于翻译——将输入文档转换为另一种格式。一个例子是**编译**。将源代码编译为机器代码的编译器首先将其解析为解析树，然后将树翻译为机器代码文档。

解析器的类型：

> 自顶向下解析器：语法的高级结构并尝试匹配其中一个
>
> 自底向上解析器：从低级规则开始，直到满足高级规则

- HTML解析器

  HTML 解析器的工作是将 HTML 标记解析为解析树。

  HTML 不容易被解析，传统的解析器无法解析，因为它的语法不是上下文无关的语法，XML 解析器也无法解析。

  原因是：

  1. 语言的宽容性质。
  2. 浏览器具有传统的容错能力以支持众所周知的无效 HTML 案例这一事实。
  3. 可重入中的解析过程。通常源在解析过程中不会改变，但在 HTML 中，包含“document.write”的脚本标签可以添加额外的标记，因此解析过程实际上会修改输入。

  解析算法在 HTML5 规范中包括两个阶段：

  > 标记化
  >
  > 树构造

  编写格式良好的 HTML。

- CSS解析

  与 HTML 不同，CSS 是一种上下文无关语法

- 解析脚本

- 处理脚本和样式表的顺序

  网络模型是同步的。作者希望在解析器到达 标记时立即解析和执行脚本。文档的解析会暂停，直到脚本被执行。如果脚本是外部的，则必须首先从网络中获取资源——这也是同步完成的，解析会暂停，直到获取资源。这是多年来的模型，也在 HTML 4 和 5 规范中指定。作者可以将脚本标记为“**延迟**”，因此它不会停止文档解析并在解析后执行。**HTML5 添加了一个将脚本标记为异步的选项，因此它将由不同的线程解析和执行。**

  推测解析：在执行脚本时，另一个线程解析文档的其余部分并找出需要从网络加载的其他资源并加载它们。推测解析器不会修改 DOM 树并将其留给主解析器，它只解析对外部资源的引用，如外部脚本、样式表和图像。

  从概念上看，由于样式表不会更改 DOM 树，因此没有理由等待它们并停止文档解析。但是，在文档解析阶段要求样式信息的脚本存在问题。如果样式还没有加载和解析，脚本会得到错误的答案，显然这会导致很多问题。这似乎是一个边缘情况，但很常见。当样式表仍在加载和解析时，Firefox 会阻止所有脚本。Webkit 仅在脚本尝试访问可能受卸载样式表影响的某些样式属性时才阻止脚本。

- 渲染树构造

  在构建 DOM 树时，浏览器会构建另一棵树，即渲染树。该树是按显示顺序排列的视觉元素。它是文档的可视化表示。此树的目的是使内容能够以正确的顺序绘制。

- 渲染树与DOM树的关系

  渲染器对应于 DOM 元素，但关系不是一对一的。非可视 DOM 元素不会插入到渲染树中。有对应于几个视觉对象的 DOM 元素。这些通常是结构复杂的元素，不能用单个矩形来描述。例如，“select”元素有 3 个渲染器。此外，当由于一行的宽度不足而将文本分成多行时，新行将作为额外的渲染器添加。

  一些渲染对象对应一个 DOM 节点，但不在树中的同一位置。浮动和绝对定位的元素不流动，放置在树中的不同位置，并映射到真实框架。占位符框架是他们应该在的地方。

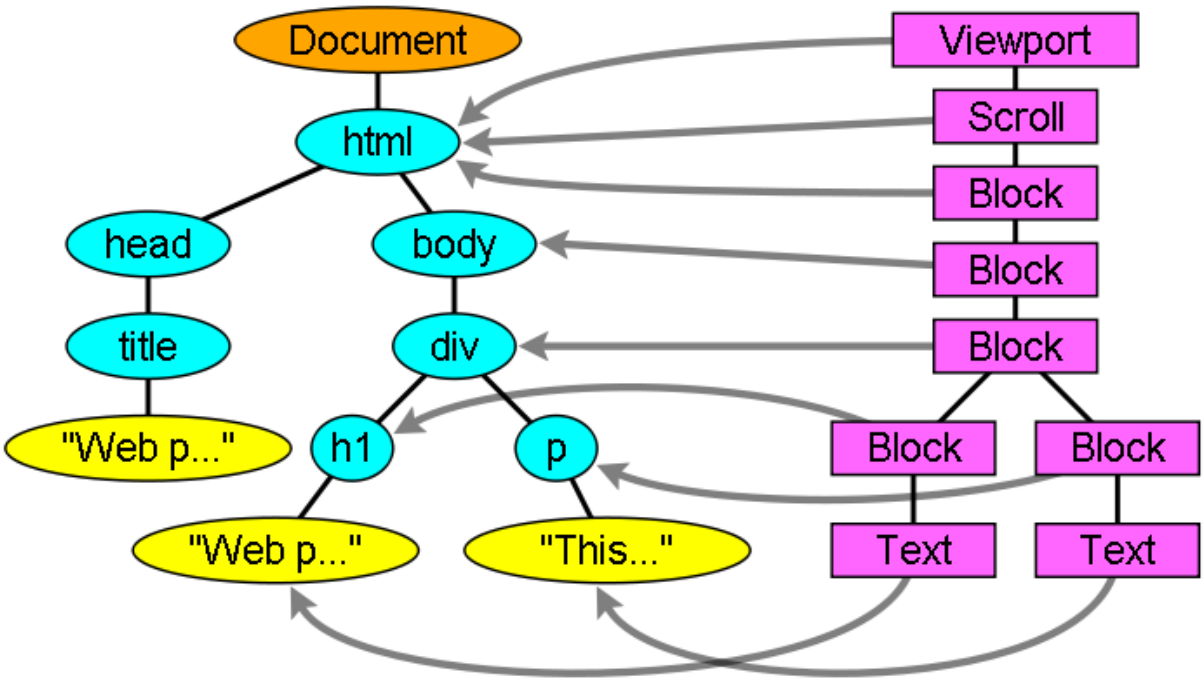

图 11：渲染树和对应的 DOM 树（3.1）。“视口”是初始包含块。在 Webkit 中，它将是“RenderView”对象。

- 构建树的流程

  在 Firefox 中，演示文稿被注册为 DOM 更新的侦听器。演示文稿将框架创建委托给“FrameConstructor”，并且构造函数解析样式

  在 Webkit 中，解析样式和创建渲染器的过程称为“附件”。每个 DOM 节点都有一个“附加”方法。附件是同步的，节点插入到 DOM 树会调用新的节点“attach”方法。

  处理 html 和 body 标签会导致渲染树根的构造。根渲染对象对应于 CSS 规范所称的包含块——包含所有其他块的最顶层块。它的尺寸是视口 - 浏览器窗口显示区域的尺寸。Firefox 将其称为 ViewPortFrame，Webkit 将其称为 RenderView。

- 风格计算

  构建渲染树需要计算每个渲染对象的视觉属性。这是通过计算每个元素的样式属性来完成的。样式包括各种来源的样式表、内联样式元素和 HTML 中的视觉属性（如“bgcolor”属性）。

  风格计算带来了一些困难：

  > 1. 样式数据是一个非常大的结构，包含许多样式属性，这可能会导致内存问题。
  > 2. 如果未对其进行优化，则为每个元素查找匹配规则可能会导致性能问题。遍历每个元素的整个规则列表以查找匹配项是一项繁重的任务。选择器可能具有复杂的结构，可能导致匹配过程从看似有希望的路径开始，但事实证明这是徒劳的，必须尝试另一条路径。
  >
  >   例如 - 这个复合选择器：

> ```
> div div div div {
> ...
> }
> ```
>
> 表示规则适用于作为 3 个 div 的后代的“
>
> ”。假设您要检查规则是否适用于给定的“
> ”元素。您选择树上的特定路径进行检查。您可能需要向上遍历节点树才能发现只有两个 div 并且该规则不适用。然后，您需要尝试树中的其他路径。
>
> 3. 应用规则涉及定义规则层次结构的相当复杂的级联规则。

浏览器解决方法：

Webkit 共享风格数据

Firefox 规则树

划分为结构

使用规则树计算样式上下文

操纵规则以轻松匹配

以正确的级联顺序应用规则

样式表级联顺序

排序规则

- 渐进的过程

  Webkit 使用一个标志来标记是否所有顶级样式表（包括@imports）都已加载。如果附加时样式未完全加载 - 使用占位符并在文档中标记，一旦加载样式表，它们将重新计算。

- 布局

当渲染器被创建并添加到树中时，它没有位置和大小。计算这些值称为布局或回流。

HTML 使用基于流的布局模型，这意味着大多数时候可以一次计算几何图形。“流中”后面的元素通常不会影响“流中”较早的元素的几何形状，因此布局可以从左到右，从上到下通过文档进行。也有例外 - 例如，HTML 表格可能需要不止一次通过

坐标系是相对于根框架的。使用顶部和左侧坐标。

布局是一个递归过程。它从根渲染器开始，它对应于 HTML 文档的元素。布局通过部分或全部帧层次递归地继续，为每个需要它的渲染器计算几何信息。

根渲染器的位置是 0,0，它的尺寸是视口 - 浏览器窗口的可见部分。

- 脏位系统

为了不为每一个小改动做一个完整的布局，浏览器使用了一个“脏位”系统。更改或添加的渲染器将其自身及其子级标记为“脏” - 需要布局。

有两个标志 - “脏”和“孩子脏”。孩子很脏意味着虽然渲染器本身可能没问题，但它至少有一个需要布局的孩子。

- 全局和增量布局
  - 全局布局（同步）
    1. 影响所有渲染器的全局样式更改，例如字体大小更改。

        2. 由于屏幕被调整大小
    - 增量布局（异步）
  - 布局过程
    1. 父渲染器确定自己的宽度
    2. 家长检查孩子并：
        1. 放置子渲染器（设置其 x 和 y）
        2. 如果需要，调用子布局（它们很脏，或者我们处于全局布局或其他原因） - 这会计算孩子的高度
    3. 父级使用子级累积高度以及边距和填充的高度来设置它自己的高度 - 这将由父级渲染器的父级使用
    4. 将其脏位设置为假
  - 优化

    当布局由“调整大小”或渲染器位置（而不是大小）的更改触发时，渲染大小将从缓存中获取并且不会重新计算。
    在某些情况下 - 仅修改子树并且布局不会从根开始。这可能发生在更改是本地的并且不影响其周围环境的情况下 - 例如插入文本字段的文本（否则每次击键都会触发从根开始的布局）。
  - 宽度计算

    渲染器的宽度是使用容器块的宽度、渲染器的样式“width”属性、边距和边框来计算的。

    clientWidth 和 clientHeight 表示对象的内部，不包括边框和滚动条。
  - 换行

    当布局中间的渲染器决定它需要中断时。它停止并传播给它需要被破坏的父级。父级将创建额外的渲染器并在它们上调用布局。
- 绘画

  在绘制阶段，遍历渲染树并调用渲染器的“paint”方法将其内容显示在屏幕上
  - 全局和增量：整个树被绘制或者是一些渲染器的变化不会影响整个树
  - 绘画顺序
    1. 背景颜色
    2. 背景图
    3. 边界
    4. 孩子们
    5. 大纲
- 动态变化

  浏览器会尝试执行尽可能少的操作以响应更改。因此，对元素颜色的更改只会导致元素的重绘。对元素位置的更改将导致元素、其子元素和可能的兄弟元素的布局和重绘。添加 DOM 节点将导致节点的布局和重绘。重大更改，例如增加“html”元素的字体大小，将导致缓存失效、依赖和重新绘制整个树。
- 渲染引擎的线程

  渲染引擎是单线程的。除了网络操作之外，几乎所有事情都发生在一个线程中。
  - 事件循环：浏览器主线程是一个事件循环。它是一个无限循环，使进程保持活力。它等待事件（如布局和绘制事件）并处理它们。
- CSS2视觉模型

- 画布：浏览器绘制内容的位置。对于空间的每个维度，画布都是无限的，但浏览器会根据视口的维度选择初始宽度。
- CSS盒子模型

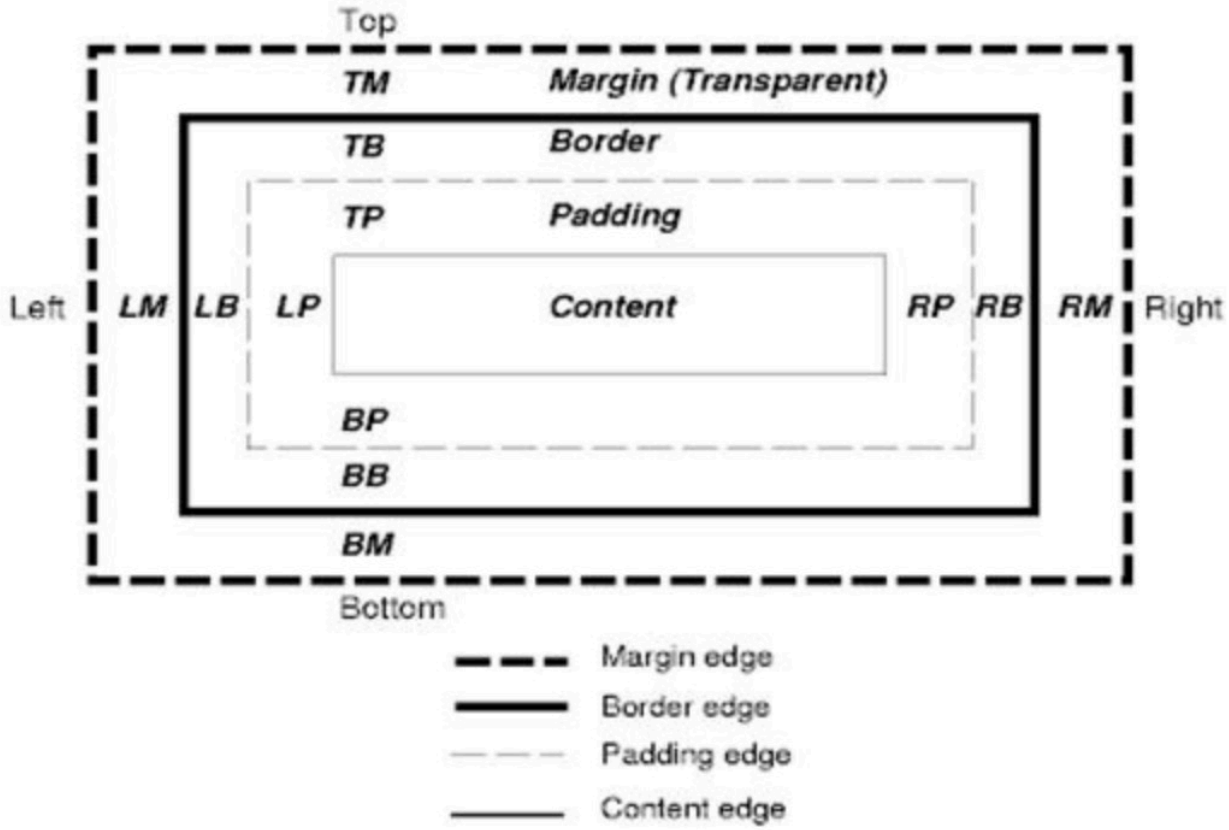

图 14：CSS2 盒子模型

- 定位方案
    1. 正常 - 对象根据其在文档中的位置定位 - 这意味着它在渲染树中的位置就像它在 dom 树中的位置，并根据其框类型和尺寸进行布局
    2. 浮动 - 对象首先像正常流程一样布局，然后尽可能向左或向右移动
    3. 绝对 - 对象在渲染树中的放置与其在 DOM 树中的位置不同
    - 定位方案由“position”属性和“float”属性设置。
- 盒子类型：Block、Inline
- 定位：相对定位、绝对定位、固定定位
- 分层表示：它由 z-index CSS 属性指定。它代表盒子的第三维，它沿着“z 轴”的位置。

3. 小结

这部分内容是和《WebKit技术内幕》穿插着看的。通过这篇文章可以对浏览器的功能有一个全局的概念，带着这些知识再去回顾《WebKit技术内幕》的内容，会更加清晰明了。