[**从输入URL 到页面加载完成的过程**](http://www.cnblogs.com/Arlar/p/5860156.html)

1. 域名解析：若是域名，通过DNS解析成IP地址
2. 建立TCP连接：浏览器根据IP地址与服务器建立socket 连接；
3. 发起HTTP请求：浏览器与服务器通信： 浏览器请求，服务器处理请求；
4. 浏览器渲染页面

**TCP/IP是互联网相关的各类协议族的总称**

**TCP/IP分为4层：应用层、传输层、网络层、链路层。**

DNS（Domain Name System）位于应用层，提供域名和IP地址之间的解析服务。

**URI和URL**

URI（Uniform Resource Identifier）：统一资源标识符。

URL（Uniform Resoure Locator）：统一资源定位符，通俗的说法是网址。

URI表示**某一互联网资源**，而URL表示**资源地点**，所以URL是URI的子集，下面是几个URI资源。

**HTML中href、src区别**

href是Hypertext Reference的缩写，表示超文本引用。用来建立当前元素和文档之间的链接。（link、a）

src是source的缩写，src指向外部资源的位置，表示引入这个资源并替换当前元素。（img、script、iframe...）

**同步和异步的区别**

**浏览器的内核**

IE-Trident

Edge-EdgeHTML

firefox-Gecko

Safari-Webkit

Chrome-Webkit-> Chromium(v8)

Opera:Presto-> Chromium(v8)

脏检查

在angular中，他没有办法判断你的数据是否做了更改， 所以它设置了一些条件，当你触发了这些条件之后，它就执行一个检测来遍历所有的数据，对比你更改了地方，然后执行变化。这个检查很不科学。而且效率不高，有很多多余的地方，所以官方称为脏检查。脏

检查即一种不关心你如何以及何时改变的数据， 只关心在**特定的检查阶段**数据是否改变的数据监听技术.

线程和进程的关系

进程：操作系统分配的占有CPU资源的最小单位。拥有独立的地址空间。

线程：安排CPU执行的最小单位。同一个进程下的所有线程，共享进程的地址空间。

简单讲，计算机就像工厂，进程是个大车间，计算机内部有很多个这样的大车间。线程是工人，每一个车间里的工人至少有一个。

关系：进程中包含着至少一个线程。在进程创建之初，就会包含一个线程，这个线程会根据需要，调用系统库函数去创建其他线程。但需要注意的是，这些线程之间是没有层级关系的，他们之间协同完成工作。在整个进程完成工作之后，其中的线程会被销毁，释放资源。

单线程：单线程就是一个进程中只有一个线程。程序顺序执行，前面的执行完，才会继续执行后面的程序。  
多线程：多线程就是一个进程中只有多个线程。在进程内部进行线程间的切换，由于每个线程执行的时间片很短，所以在感觉上是并行的。

虽然JavaScript是单线程，但是浏览器是事件驱动的、异步的、多线程的。

浏览器至少有四个线程（不同浏览器会有差异）： js引擎线程、界面渲染线程、事件触发线程、http请求线程。

延时器：js在执行到延时函数时，会触发浏览器的定时器，到设置时间，浏览器再将这个函数放入执行的函数队列，再由JavaScript引擎执行。都是在浏览器空闲了才会执行。

AJAX的异步：在调用AJAX的时候，浏览器会开辟一个新的线程，去处理这个请求，得到响应后，如果这个请求有回调，会将这个回调再放入事件队列中。再由JavaScript引擎执行。

JavaScript的运行机制：javascript是单线程。单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。任务可以分成两种，一种是同步任务（synchronous），另一种是异步任务（asynchronous）。同步任务指的是，在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务；异步任务指的是，不进入主线程、而进入"任务队列"（task queue）的任务，只有等主线程任务执行完毕（空闲了），"任务队列"开始通知主线程，请求执行任务，该任务才会进入主线程执行。

只要主线程空了，就会去读取"任务队列"，这就是JavaScript的运行机制。

GET与POST(GET，POST，PUT，DELETE，HEAD)：

1. Get是向服务器发获取数据的一种请求，而Post是向服务器提交数据的一种请求。
2. get是把 参数数据队列 加到提交表单的ACTION属性所指的URL中，值和表单内各个字段一一对应，在URL中可以看到。post是通过HTTP post机制，将表单内各个字段与其内容放置在HTML HEADER内一起传送到ACTION属性所指的URL地址。用户看不到这个过程。
3. 对于get方式，[服务器端](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8%E7%AB%AF&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9mvD1mWmYP1bdPvw-m1w-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErjTdn1RLrH6)用Request.QueryString获取变量的值，对于post方式，[服务器端](https://www.baidu.com/s?wd=%E6%9C%8D%E5%8A%A1%E5%99%A8%E7%AB%AF&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9mvD1mWmYP1bdPvw-m1w-0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3ErjTdn1RLrH6)用Request.Form获取提交的数据。
4. get传送的数据量较小，不能大于2KB。post传送的数据量较大，一般被默认为不受限制，IIS5中为100KB
5. get安全性非常低，post安全性较高。但get执行效率却比Post方法好。
6. 建议：  
   1、get方式的安全性较Post方式要差些，包含机密信息的话，建议用Post数据提交方式；  
   2、在做数据查询时，建议用Get方式；而在做数据添加、修改或删除时，建议用Post方式；

## 内存泄漏（memory leak）

不再用到的内存，没有及时释放，就叫做内存泄漏。

有些语言（比如 C 语言）必须手动释放内存，大多数语言提供自动内存管理，减轻程序员的负担，这被称为"垃圾回收机制"（garbage collector）。

**垃圾收集器会定期（周期性）**找出那些不在继续使用的变量，然后释放其内存。

通常情况下有两种实现方式：标记清除和引用计数

标记清除：js中最常用的垃圾回收方式就是标记清除。当变量进入环境时，就将这个变量标记为“进入环境”。而当变量离开环境时，则将其标记为“离开环境”。 最后，垃圾回收器销毁那些带标记的值并回收它们所占用的内存空间。到目前为止，IE、Firefox、Opera、Chrome、Safari的js实现使用的都是标记清除的垃圾回收策略或类似的策略。

引用计数：引用计数的含义是跟踪记录每个值被引用的次数。当这个值的引用次数变成0时，则说明没有办法再访问这个值了，因而就可以将其占用的内存空间回收回来。这样，当垃圾回收器下次再运行时，它就会释放那些引用次数为0的值所占用的内存。

老IE（7，8）下易出现内存泄漏的原因：

IE中有一部分对象并不是原生js对象。例如，其内存泄露DOM和BOM中的对象就是使用C++以COM对象的形式实现的，而COM对象的垃圾回收机制采用的就是引用计数策略。只要在IE中涉及COM对象，就会存在循环引用的问题。

循环引用：

obj1={},obj2 = {};

obj1.a=obj2;obj2.a=obj1;

循环引用就像打了一个死结，他们的引用计数永远不会等于0，即使它们不再被使用到了，也不会被gc回收。

在ie上最常见的造成内存泄漏的例子：

~*function* () {

*var* obj = document.querySelector("#div");

obj.onclick = *function* () {

//...

}

}()

解决方法：

~*function* () {

*var* obj = document.querySelector("#div");

obj.onclick = *function* () {

//...

}

obj = null;

}()

# 内存泄漏和内存溢出区别:

# 内存泄漏: 内存泄露是指你的应用使用资源之后没有及时释放，导致应用内存中持有了不需要的资源，这是一种状态描述;

# 内存溢出: 内存溢出是指浏览器的内存已经不能满足正常使用了，堆栈已经超过系统设置的最大值，进而导致崩溃，这事一种结果描述。

# 通常都是由于内存泄露导致堆栈内存不断增大，从而引发内存溢出。

# 事件流

# 事件流描述的是从页面中接收**事件**的顺序,也可理解为**事件**在DOM结构中传播的顺序。

# DOM2级的事件

# 而DOM2级的事件规定了事件流包含三个阶段包括： **1：事件捕获， 2：处于目标阶段， 3：事件冒泡阶段**（IE8以及更早版本不支持DOM事件流）;

事件委托

事件委托是基于事件冒泡的，在父元素上侦听子元素或子孙元素的事件，从而减少性能开销。

###### 前端性能优化方案:

1. 减少http请求，减少请求数，降低请求量
2. Css雪碧图
3. 文件打包压缩，包括CSS、JavaScript、图片的压缩
4. 图片懒加载
5. 二进制的矢量图标
6. 减少对DOM的操作
7. 减少重排与重绘
8. CSS放在页面最上部，javascript放在页面最下面
9. 减少cookie传输
10. CDN加速，即内容分发网络。资源服务器。CDN缓存的一般是静态资源，如图片、文件、CSS、script脚本、静态网页等，但是这些文件访问频度很高，将其缓存在CDN可极大改善网页的打开速度。
11. 避免使用 eval和arguments
12. http缓存

广义的跨域

跨域是指一个域下的文档或脚本试图去请求另一个域下的资源，这里跨域是广义的。

其实我们通常所说的跨域是狭义的，是由浏览器同源策略限制的一类请求场景。

同源策略

同源策略（Same origin policy，简称SOP）是一种约定，由Netscape公司1995年引入浏览器，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，浏览器很容易受到XSS、CSFR等攻击。所谓同源是指"协议+域名+端口"三者相同，即使两个不同的域名指向同一个ip地址或者域名和域名对应的ip，也属于“非同源”。

localhost和127.0.0.1虽然都指向本机，但也属于跨域。

同源策略限制以下几种行为：

1.) Cookie、LocalStorage 和 IndexDB 无法读取

2.) DOM 和 Js对象无法获得

3.) AJAX 请求不能发送

跨域解决方案

1、 通过jsonp跨域

2、 document.domain + iframe跨域（应用于主域相同而子域不同）

3、 location.hash + iframe

4、 window.name + iframe跨域

5、 postMessage跨域

6、 跨域资源共享（CORS）

7、 nginx代理跨域

8、 nodejs中间件代理跨域

9、 WebSocket协议跨域

重绘（redraw或repaint）,重排（重构/回流/reflow）

重绘：指一个元素外观的改变所触发的浏览器行为，浏览器会根据元素的新属性重新绘制，使元素呈现新的外观。

触发重绘的条件：改变元素外观属性。如：color，background-color等。

重排：当渲染树中的一部分(或全部)因为元素的规模尺寸、布局、显示隐藏等改变而需要重新构建, 这就称为重排(**reflow**)。每个页面至少需要一次重排，就是在页面第一次加载的时候。

触发重排的条件：任何页面布局（位置，显隐）和几何属性（尺寸）的改变都会触发重排，如width，height，display:none等

以下都会触发重排：

1、页面渲染初始化；(无法避免)

　　2、添加或删除可见的DOM元素；

　　3、元素位置的改变，或者使用动画；

　　4、元素尺寸的改变——大小，外边距，边框；

　　5、浏览器窗口尺寸的变化（resize事件发生时）；

　　6、填充内容的改变，比如文本的改变或图片大小改变而引起的计算值宽度和高度的改变；

　 7、读取某些元素属性：（offsetLeft/Top/Height/Width,　clientTop/Left/Width/Height,　scrollTop/Left/Width/Height,　width/height,　getComputedStyle(),　currentStyle(IE)　)

重绘和重排的关系：

先重排，完成后再重绘。

重排必定引发重绘，但重绘不一定引发重排。

如何减少重排:

（1）不要一条条的修改DOM样式，改变className来“统一修改”（注：现代浏览器会进行增量异步reflow，即“攒够”一些修改后再统一修改）

（2）把dom离线修改，比如将其至于内存或设置display:none。

（3）为html动画原件使用position:absolute布局或fixed布局，这样不会引起重排和重绘。

（4）不要使用table布局，因为稍加改动就会导致table的重新布局。

（5）将offsetTop等属性缓存成局部变量。

http状态码

1XX：表示可续发请求

2XX：表示成功

\* 202成功

\* 204成功 不返回实体主体

\* 206成功 执行了一个范围请求

3XX：表示重定向

\* 301永久重定向，增加SEO排名

\* 302临时重定向 禁止POST变为GET

\* 303另外一个URI

\* 304判断是否要更新缓存 请求头部携带if-modified-since自从上次更新距这次多久

\* 307临时重定向

4XX：表示客户端错误

\* 400客户端语法错误

\* 401请求未经授权

\* 403服务器拒绝服务

\* 404请求资源不存在

5XX：服务端错误

\* 500不可预期的错误

\* 503此时不能提供服务 稍后恢复正常

常见的Web攻击有哪些？

（1）跨站脚本攻击（XSS）

（2）分布式拒绝服务（DDOS）

基本方法是利用合理的请求占用服务器的大量资源，使正常用户无法得到服务器的相应。

（3）跨站请求伪造攻击（CSRF）

攻击者通过各种方法伪造一个请求，模仿用户提交表单的行为，从而达到修改用户的数据，或者执行特定任务的目的。解决方法是尽量使用post请求。

（4）sql注入

浏览器是如何渲染页面的？

1. 解析html\xhtml\svg，生成一个dom树
2. 解析css文件生成css规则树，解析js以获得其对html和css的影响
3. 解析完成后，浏览器会根据dom树和css规则树来生成渲染树。
4. 进行dom元素布局 (layout/reflow)，定义元素的坐标。
5. 遍历渲染树进行绘制，即绘制（paint/repaint）。

cookie和session的机制是什么？有什么区别？

会话跟踪是用来跟踪用户的整个会话，它可以弥补http协议无状态的不足。常用的会话跟踪技术是cookie与session。cookie通过在客户端记录信息确定用户身份，session通过在服务器端记录信息确定用户身份。

HTTP：

超文本传输协议，是基于TCP/IP协议的应用层协议。主要规定了客户端和服务器之间的通信格式，默认使用80端口。

HTTP是无状态协议，每次请求都必须附上所有信息。

HTTPS：

是以安全为目标的HTTP通道，简单讲是HTTP的安全版。

HTTPS协议是由SSL（TLS）+HTTP协议构建的可进行加密传输、身份认证的网络协议。

HTTPS和HTTP的区别主要为以下四点：

一、https协议需要到ca申请证书，一般免费证书很少，需要交费。

二、http是超文本传输协议，信息是明文传输，https 则是具有安全性的ssl加密传输协议。

三、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

四、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

HTTP/2：

致力于突破http1.1的性能限制，改进传输性能，实现低延迟和高吞吐量。但核心概念上保持不变。

升级 HTTP/2 的前提条件是什么？

答：由于现在所有支持 HTTP/2 的浏览器都强制只使用 TLS(https) 连接，所以：获取证书，并且让服务器支持 https 是必须的先决条件。

对HTTP1.1有4个改进：

二进制分帧、多路复用、头信息压缩、服务器推送。

二进制分帧：

HTTP/1.1 版的头信息必须是文本（ASCII编码），数据体可以是文本或是二进制。HTTP/2 则是一个彻底的二进制协议，头信息和数据体都是二进制，并且统称为"帧"（frame）：头信息帧和数据帧。文本的表现形式有多样性，二进制则不同，只认0和1的组合，所以HTTP2的协议解析决定采用二进制格式，实现方便且健壮。

既然又要保证HTTP的各种动词，方法，首部都不受影响，那就需要在应用层(HTTP2.0)和传输层(TCP or UDP)之间增加一个二进制分帧层。

多路复用（连接共享）：

HTTP/2引入连接共享机制，允许一个连接里，同时发送多个请求和接受多个回应，因此 HTTP/2 可以很容易的去实现多流并行而不用依赖建立多个 TCP 连接。从而提高了性能。

头信息压缩：

HTTP是无状态协议，每次请求都必须附上所有信息。请求的很多字段都是重复的，只需发送一次即可，但每次请求都必须附带，这会浪费性能，影响速度。HTTP/2对这一点做了优化，引入了头信息压缩机制（header compression）。一方面，对头信息压缩后再发送；另一方面，客户端和服务端同时维护一张头信息表，所有字段都会存入这个表，这样后面每次传输只需要传输表里面的索引Id就行，通过索引ID就可以知道表头的值。

服务器推送（缓存推送）：

HTTP/2允许服务器未经请求，主动向客户端发送资源，这叫做服务器推送。

服务器可以对一个客户端请求发送多个响应，服务器可以预期到客户端请求网页后，很可能会再请求静态资源，所以就主动把这些静态资源随着网页一起发给客户端了

-