1、浏览器会开启一个线程来处理这个请求，对 URL 分析判断如果是 http 协议就按照 Web 方式来处理;

2、调用浏览器内核中的对应方法，比如 WebView 中的 loadUrl 方法;

3、通过DNS解析获取网址的IP地址，设置 UA 等信息发出第二个GET请求;

4、进行HTTP协议会话，客户端发送报头(请求报头);

5、进入到web服务器上的 Web Server，如 Apache、Tomcat、Node.JS 等服务器;

6、进入部署好的后端应用，如 PHP、Java、JavaScript、Python 等，找到对应的请求处理;

7、处理结束回馈报头，此处如果浏览器访问过，缓存上有对应资源，会与服务器最后修改时间对比，一致则返回304;

8、浏览器开始下载html文档(响应报头，状态码200)，同时使用缓存;

9、文档树建立，根据标记请求所需指定MIME类型的文件（比如css、js）,同时设置了cookie;

10、页面开始渲染DOM，JS根据DOM API操作DOM,执行事件绑定等，页面显示完成。

  
前端与后台通信的发展阶段

第一阶段：Form

web应用想要与服务器交互，必须提交一个表单（form）。服务器接收并处理该表单，然后返回一个全新的页面。 缺点：前后两个页面需要更新的数据可能很少，这个过程可能传输了很多之前那个页面已经有的数据，浪费了带宽；为了返回整个页面，服务器可能需要为页面的其他内容构造数据，浪费服务器资源。

第二阶段：Ajax（Asynchronous JavaScript and XML）

由JS脚本发送http请求，服务器只返回页面需要更新的那部分数据，JS根据返回的数据刷新页面的局部内容，而不是刷新整个页面。 优点：网络传输小、服务器负担小 缺点：没有解决对实时性的友好支持（比如聊天室或客服系统），如果需要持续地从服务器获取数据，需要不断请求

第三阶段：Comet

一种hack技术，基于http长连接的“服务器推”的技术。实现方式有很多种，比如基于Ajax的长轮询（long-polling）、封装好的Comet4J等。 比较简单的是基于Ajax的长轮询：web应用发出Ajax请求，服务器收到请求后，如果有数据就即刻返回，如果没数据，会等待一段时间（比如定时20秒），在等待的时间内，如果有数据就返回，如果一直没数据，则timeout的时候返回。web应用收到服务器的返回后，处理数据、更新页面，再继续请求。服务器在发送完数据到接收到一个新请求之前，如果有数据需要发送，则先缓存起来，等待接到下一次请求后再发送。 优点：基本解决了实时性的问题 缺点：服务器需要阻塞线程等待数据，如果连接请求多的话，得考虑使用复杂的技术释放线程给其他请求使用，不然可能会拖死服务器。需要客户端和服务器端协调好才能正常工作。

第四阶段：Websocket

html5标准中的一种全双工通信规范，基于TCP协议，与http协议兼容，但不是http协议的一部分。 web应用（客户端）先发起一个Handshake请求，服务器应答之后，就可以双向交换数据了。 通信的两方都可以发起关闭的请求（通过发送一个Closing Handshake），对方收到请求后，回复一个相同的数据包，就关闭通信通道了。 优点：数据传输量极少、实时性好。 --------------------- 本文来自 onesilver 的CSDN 博客 ，全文地址请点击：https://blog.csdn.net/onesilver/article/details/77969091?utm\_source=copy

## .对于mvvm的理解

MVVM 是 Model-View-ViewModel 的缩写。  
Model代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑。  
View 代表UI 组件，它负责将数据模型转化成UI 展现出来。  
ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步View 和 Model的对象，连接Model和View。  
在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。  
ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

## 38.vue生命周期

beforeUpdate（更新前） 在数据更新之前调用，发生在虚拟DOM重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。  
updated（更新后） 在由于数据更改导致的虚拟DOM重新渲染和打补丁之后调用。调用时，组件DOM已经更新，所以可以执行依赖于DOM的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。  
beforeDestroy（销毁前） 在实例销毁之前调用。实例仍然完全可用。  
destroyed（销毁后） 在实例销毁之后调用。调用后，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

不是用变量访问对象中的属性时，[ ]中的属性 要使用引号包起来。

用变量访问对象的属性时，[ ]中的变量不用加引号。

第一阶段：Form

web应用想要与服务器交互，必须提交一个表单（form）。服务器接收并处理该表单，然后返回一个全新的页面。 缺点：前后两个页面需要更新的数据可能很少，这个过程可能传输了很多之前那个页面已经有的数据，浪费了带宽；为了返回整个页面，服务器可能需要为页面的其他内容构造数据，浪费服务器资源。

第二阶段：Ajax（Asynchronous JavaScript and XML）

由JS脚本发送http请求，服务器只返回页面需要更新的那部分数据，JS根据返回的数据刷新页面的局部内容，而不是刷新整个页面。 优点：网络传输小、服务器负担小 缺点：没有解决对实时性的友好支持（比如聊天室或客服系统），如果需要持续地从服务器获取数据，需要不断请求

第三阶段：Comet

一种hack技术，基于http长连接的“服务器推”的技术。实现方式有很多种，比如基于Ajax的长轮询（long-polling）、封装好的Comet4J等。 比较简单的是基于Ajax的长轮询：web应用发出Ajax请求，服务器收到请求后，如果有数据就即刻返回，如果没数据，会等待一段时间（比如定时20秒），在等待的时间内，如果有数据就返回，如果一直没数据，则timeout的时候返回。web应用收到服务器的返回后，处理数据、更新页面，再继续请求。服务器在发送完数据到接收到一个新请求之前，如果有数据需要发送，则先缓存起来，等待接到下一次请求后再发送。 优点：基本解决了实时性的问题 缺点：服务器需要阻塞线程等待数据，如果连接请求多的话，得考虑使用复杂的技术释放线程给其他请求使用，不然可能会拖死服务器。需要客户端和服务器端协调好才能正常工作。

第四阶段：Websocket

html5标准中的一种全双工通信规范，基于TCP协议，与http协议兼容，但不是http协议的一部分。 web应用（客户端）先发起一个Handshake请求，服务器应答之后，就可以双向交换数据了。 通信的两方都可以发起关闭的请求（通过发送一个Closing Handshake），对方收到请求后，回复一个相同的数据包，就关闭通信通道了。 优点：数据传输量极少、实时性好。

#### 构造函数

通过原型这种机制，js对象从其他对象中继承功能特性

目标 ：

理解 js对象的原型 ，原型链。

如何向prototype属性添加新的方法。

1. JavaScript 常被描述为一种**基于原型的语言 (prototype-based language)**——每个对象拥有一个**原型对象**，对象以其原型为模板、从原型继承方法和属性。原型对象也可能拥有原型，并从中继承方法和属性，一层一层、以此类推。这种关系常被称为**原型链 (prototype chain)**，它解释了为何一个对象会拥有定义在其他对象中的属性和方法。

准确地说，这些属性和方法定义在Object的构造器函数(constructor functions)之上的prototype属性上，而非对象实例本身。

在传统的 OOP 中，首先定义“类”，此后创建对象实例时，类中定义的所有属性和方法都被复制到实例中。在 JavaScript 中并不如此复制——而是在对象实例和它的构造器之间建立一个链接（它是\_\_proto\_\_属性，是从构造函数的prototype属性派生的），之后通过上溯原型链，在构造器中找到这些属性和方法。

函数

1. 由于JavaScript的函数可以嵌套，此时，内部函数可以访问外部函数定义的变量，反过来则不行：
2. JavaScript的函数在查找变量时从自身函数定义开始，从“内”向“外”查找。如果内部函数定义了与外部函数重名的变量，则内部函数的变量将“屏蔽”外部函数的变量。

### 变量提升JavaScript的函数定义有个特点，它会先扫描整个函数体的语句，把所有申明的变量“提升”到函数顶部：JavaScript引擎自动提升了变量y的声明，但不会提升变量y的赋值。由于JavaScript的这一怪异的“特性”，我们在函数内部定义变量时，请严格遵守“在函数内部首先申明所有变量”这一规则。最常见的做法是用一个var申明函数内部用到的所有变量：

4、的JavaScript文件如果使用了相同的全局变量，或者定义了相同名字的顶层函数，都会造成命名冲突，并且很难被发现。

减少冲突的一个方法是把自己的所有变量和函数全部绑定到一个全局变量中。

表严肃 使用

;(function(){

}

)

();

### 5、局部作用域

由于JavaScript的变量作用域实际上是函数内部，我们在for循环等语句块中是无法定义具有局部作用域的变量的：

**function** foo() {

**for** (**var** i=0; i<100; i++) {

*//*

}

i += 100; *// 仍然可以引用变量i*

}

为了解决块级作用域，ES6引入了新的关键字let，用let替代var可以申明一个块级作用域的变量：

'use strict';

**function** foo() {

**var** sum = 0;

**for** (**let** i=0; i<100; i++) {

sum += i;

}

*// SyntaxError:*

i += 1;

}

### 常量

由于var和let申明的是变量，如果要申明一个常量，在ES6之前是不行的，我们通常用全部大写的变量来表示“这是一个常量，不要修改它的值”：

**var** PI = 3.14;

ES6标准引入了新的关键字const来定义常量，const与let都具有块级作用域：

### 解构赋值

从ES6开始，JavaScript引入了解构赋值，可以同时对一组变量进行赋值

var [x, y, z] = ['hello', 'JavaScript', 'ES6'];

This

JavaScript的函数内部如果调用了this，那么这个this到底指向谁？

答案是，视情况而定！

如果以对象的方法形式调用，比如xiaoming.age()，该函数的this指向被调用的对象，也就是xiaoming，这是符合我们预期的。

如果单独调用函数，比如getAge()，此时，该函数的this指向全局对象，也就是window。

坑爹啊！

### 小心你的return语句

前面我们讲到了JavaScript引擎有一个在行末自动添加分号的机制，这可能让你栽到return语句的一个大坑：

**function** foo() {

**return** { name: 'foo' };

}

foo(); *// { name: 'foo' }*

如果把return语句拆成两行：

**function** foo() {

**return**

{ name: 'foo' };

}

foo(); *// undefined*

要小心了，由于JavaScript引擎在行末自动添加分号的机制，上面的代码实际上变成了：

**function** foo() {

**return**; *// 自动添加了分号，相当于return undefined;*

{ name: 'foo' }; *// 这行语句已经没法执行到了*

}

所以正确的多行写法是：

**function** foo() {

**return** { *// 这里不会自动加分号，因为{表示语句尚未结束*

name: 'foo'

};

}

## 对象

总结一下，有这么几条规则需要遵守：

不要使用new Number()、new Boolean()、new String()创建包装对象；

用parseInt()或parseFloat()来转换任意类型到number；

用String()来转换任意类型到string，或者直接调用某个对象的toString()方法；

通常不必把任意类型转换为boolean再判断，因为可以直接写if (myVar) {...}；

typeof操作符可以判断出number、boolean、string、function和undefined；

判断Array要使用Array.isArray(arr)；

判断null请使用myVar === null；

判断某个全局变量是否存在用typeof window.myVar === 'undefined'；

函数内部判断某个变量是否存在用typeof myVar === 'undefined'。

最后有细心的同学指出，任何对象都有toString()方法吗？null和undefined就没有！确实如此，这两个特殊值要除外，虽然null还伪装成了object类型。

更细心的同学指出，number对象调用toString()报SyntaxError：

事实上，一种极其常见的对象定义模式是，在构造器（函数体）中定义属性、在 prototype 属性上定义方法。如此，构造器只包含属性定义，而方法则分装在不同的代码块，代码更具可读性。例如：

// 构造器及其属性定义

function Test(a,b,c,d) {

// 属性定义};

// 定义第一个方法

Test.prototype.x = function () { ... }

// 定义第二个方法

Test.prototype.y = function () { ... }

// 等等……