**1. JavaScript**

JavaScript（简称js）是一种主要运行于浏览器中的弱类型的动态脚本语言，可以用来实现网页上的一些高级功能，如数据验证处理、页面动态效果、定时任务、与用户交互、发送/接收服务器端数据等等。

动态语言指的是程序运行时可以改变结构，主要体现在：  
① js中的变量在声明的时候不需要指定类型，其实际类型由程序运行中的赋值决定，在运行过程中变量的类型也可以改变。**注：这一点是动态语言的特征，并不是弱类型语言的特征，之前的回答有误。**  
② 函数可变。js允许在运行过程中使用eval动态执行字符串里的命令，也可以通过new Function等方式由字符串动态构造函数，函数可以被创建、修改、删除，可以从已有函数构造出新函数，等等。  
③ 对象的成员可变，可以动态添加、删除成员属性或成员方法。

弱类型指的是js中的变量在参与运算的时候可以根据实际需要动态转换类型。与之相对应的是强类型语言——变量一般不允许自动转换类型（某些强类型语言的字符串连接操作除外），如果参与运算、调用时不符合要求的类型，则会

**2. jQuery**

jQuery是js的一个工具库，由John Resig在2006年发布。  
j代表JavaScript，query是“查询”的意思。也就是说，这个库的意图是基于JavaScript的查询。  
查询的目标是什么？答案是DOM（文档对象模型）结构中的Node（节点）。一个网页就是一个html文档，而网页上的所有内容都是节点，包括文档节点、元素节点、文本节点、注释节点、属性节点等等。而jQuery的查询最主要针对的是元素节点，如段落（p）、锚点（a）、表格（table）等，只有少数方法可以处理文本节点与注释节点。同时jQuery还可以用attr方法方便地对元素节点的属性进行读取/设置。

在jQuery出现之前，在js程序中获取元素节点比较麻烦，例如获取id为elem1的节点：

document.getElementById('elem1')

或者是获取页面上的所有checkbox元素，首先需要获取input类型的元素：

document.getElementsByTagName('input')

然后对获得的元素列表进行for循环处理，逐个判断其类型是否为checkbox。  
如果有更多元化的查询要求，则对应的js代码也会相当复杂。虽然有一些库可以解决这方面的需求，但强大程度、易用性等方面都不太理想。

John Resig发现了一个盲点——css样式应用到页面上的元素时，是有一套规则的，即css选择器，浏览器可以通过css选择器找到匹配的元素并将指定的样式应用到这些元素上。也就是说，通过css选择器可以有效地进行元素查找定位，但它最初只被用于样式领域。于是，John Resig根据css选择器编写了jQuery选择器，并对选择器的规则进行了扩充，从而让元素查找变得非常方便。例如，上面2个例子用jQuery可以写为：

$('#elem1')

与

$(":checkbox")

同时，jQuery还有一个核心思想——链式操作，例如：

$('div.con')

.height(100)

.show();

这样的连续调用可以让代码书写更加简洁，也就是jQuery自己的口号：write less, do more。

此外，jQuery还提供了浏览器兼容、样式读写、事件绑定与执行、动画等特性，后来又加入了ajax、promise等，再加上方便的插件编写机制，对整个js的生态圈产生了重大的影响，可以说是js历史上影响力最大的一个库。其中选择器引擎后来被单独剥离出来成为sizzle，供其他的js库调用。这部分的工作还影响了官方，在jQuery成功之后，浏览器才有了querySelector与querySelectorAll方法。时至今日，虽然有了querySelector与querySelectorAll，但jQuery的选择器仍然有少部分特性是前2者所无法替代的。

**3. ajax**

ajax全称Asynchronous JavaScript and XML（异步的JavaScript与XML），是**网页无需刷新页面、使用js与服务器进行交互的一种技术。**

有时候会有这样一种需求：只希望更改页面上的一个区域。然而在从前的技术框架内只能刷新整个页面，带来的后果是：①需要重新传输整个页面，服务器端与客户端的流量消耗都会比较大；②如果是动态页，服务器端需要重新生成整个页面，即使是那些客户原本不想要刷新的区域，增大了服务器的负担。

Google的Jesse James Garrett在2005年初发表了一篇文章，提供了解决这种需求的技术方案，也就是ajax。实际上这是一种实践先行的技术，该方案的技术依赖之一XMLHTTP在1998年就已经被Microsoft开发出来了，而Google在若干年后使用这项技术开发Google Maps等产品之后，才发表了相应的文章并对其进行了命名。

ajax的基本流程可以概括为：页面上js脚本实例化一个XMLHttpRequest对象，设置好服务器端的url、必要的查询参数、回调函数之后，向服务器发出请求，服务器在处理请求之后将处理结果返回给页面，触发事先绑定的回调函数。这样，页面脚本如果想要改变一个区域的内容，只需要通过ajax向服务器获取与该区域有关的少量数据，在回调函数中将该区域的内容替换掉即可，不需要刷新整个页面。

XMLHttpRequest在发送请求的时候，有两种方式：同步与异步。同步方式是请求发出后，一直到收到服务器返回的数据为止，浏览器进程被阻塞，页面上什么事也做不了。而异步方式则不会阻塞浏览器进程，在服务端返回数据并触发回调函数之前，用户依然可以在该页面上进行其他操作。ajax的核心是异步方式，而同步方式只有在极其特殊的情况下才会被用到。

XMLHttpRequest在早期IE浏览器里是使用ActiveX来实现的，并不是浏览器自身的对象。后来其他各家浏览器也都实现了XMLHttpRequest对象，而高版本IE也把XMLHttpRequest改为了浏览器的内建对象。

**4. JSON**

JSON全称JavaScript Object Notation（js对象标记法），由Douglas Crockford在2002年发现并制定了标准。从名称上就可以看出来，JSON是基于JavaScript的，是JavaScript的一个子集。JSON是用JavaScript语法来**表示数据的一种轻量级语言。**

虽然Douglas在2002年就注册了[http://json.org](https://link.zhihu.com/?target=http%3A//json.org" \t "_blank)，并且为各种语言编写了解析与构造JSON数据的库，但在最开始的几年JSON一直没有得到足够的重视。情况一直延续到ajax的出现。

从ajax的命名中我们就可以看到，数据交换是通过XML格式进行的。在ajax刚出现的时候，绝大多数应用都是采用XML格式，也有少数使用纯文本的。但是XML格式有一个缺点，就是文档构造复杂，需要传输比较多的字节数。在这种情况下，JSON的轻便性逐渐得到重视，后来替代XML成为ajax最主要的数据传输格式。可以举个简单的例子感受一下二者的区别：

<?xml version="1.0" encoding="utf-8"?>

<root>

<article>

<title>Article Title1</title>

<content>content1</content>

</article>

<article>

<title>Article Title2</title>

<content>content2</content>

</article>

</root>

{

"article" : [

{

"title": "Article Title1",

"content": "content1"

},

{

"title": "Article Title2",

"content": "content2"

}

]

}

XML规范实际上是比较复杂的，单纯作为数据传输来说它太重了。在ajax领域中JSON取代XML的过程，是一个很好的“用脚投票”的范例。

而JSON的影响力在此后还继续扩大，有些软件将其作为配置文件的格式，有些编程语言也吸纳了JSON的优点。例如c#，在高版本里可以这样写：

Dictionary<int, string> dict = new Dictionary<int, string>{

{1, "a"},

{2, "b"}

};

但是如果c# 2.0这样写，可是会报错的。在2.0里只能写成下面这种形式：

Dictionary<int, string> dict = new Dictionary<int, string>();

dict.Add(1, "a");

dict.Add(2, "b");

等价于

Dictionary<int, string> dict = new Dictionary<int, string>();

dict[1] = "a";

dict[2] = "b";

比较一下两种写法的区别，不仅有便捷性的差距，而且前一种写法可以在声明变量的同时为变量赋值，后一种写法则不行，这会影响到类属性的初始化操作：在c# 2.0中，只能把针对Dictionary之类复杂对象的初始化代码写在函数里，而不能直接写在类属性的声明处。

感觉上是c#受了JSON（或者说js）的影响。但此处是我个人的感觉，如有错误请指出。

回到js自身，对于对象构造有两种方法：基于对象的完整写法，字面量表示法。前者如：

var obj = new Object();

obj.title = "title1";

obj.content = "content1";

而与之对应的字面量表示法则写为：

var obj = {

title: "title1",

content: "content1"

};

可以明显看出字面量表示法要简洁得多。而JSON基本就是字面量表示法的一个子集，除了强制要求键与字符串类型的值必须用双引号包起之外，它剔除了undefined、function等类型，也不包括浏览器内置对象类型（如Date、RegExp等），是基于文本的、比较纯粹的数据表示方法。所以说，Douglas是“发现”了JSON，而不是“发明”。标准的JSON不包含注释，但后来因为实际需求而出现了能够处理注释的JSON库。

**5. Node.js**

Node.js是Ryan Dahl在2009年发布的、主要用于服务器端的Javascript运行环境，也可以用于个人电脑。  
Ryan Dahl此前一直在寻找一种事件驱动型的、异步的服务器端框架，实际上，js并不是他的首选。他是在尝试了几种语言之后，才发现js的函数回调与单线程特性正好契合他的要求，于是Node.js应运而生。  
js的异步回调在ajax的部分已经提过：在调用异步方法的时候，可以将后续的处理函数作为参数传入，在调用相应的异步接口之后，程序会将线程的控制权让出，允许其他代码执行；在接口返回处理结果后，再执行后续处理函数（即回调函数）。实际上，因为js是单线程语言，回调函数并不是立刻被执行的，而是会被送入任务队列，在线程空闲、并且队列前方没有其他任务的情况下，才会被执行。

用户在向服务器提交请求的时候，如果处理比较费时，传统的服务器端框架会导致处理线程被阻塞。而js的特性使得异步任务在执行的时候让出线程的控制权，在处理完成后再进行正确的回调，从而能够获得比较好的高并发处理能力。  
js本身是一门严格的单线程语言，而Node.js为了充分发挥服务器的处理能力，在运行环境级别上增加了对于多线程的支持（child process）。但Node.js的多线程与常规的多线程有很大区别——常规语言的多线程允许多个线程共享数据，或者调用其他线程暴露出来的公开方法，而Node.js的多线程只能用消息机制进行通讯。这样，Node.js就规避了常规多线程的数据同步、线程锁（线程同步/互斥）等复杂问题，规避了一些潜在风险。

Node.js使用的V8引擎实际上就是Google的Chrome浏览器使用的Javascript引擎（因为V8引擎是开源的），并进行了模块扩展。例如遵循CommonJS标准的模块定义，适合服务器需求的多线程、集群、HTTP/HTTPS，文件系统，等等。Node.js中的很多方法都同时提供了异步版本与同步版本，从函数的命名上可以简单区分。  
得益于其模块特性，Node.js的模块扩展变得相当方便，用于Node.js包管理的npm得到了广泛的使用，但也曾经引起“是否过度使用依赖包”的争论。

Node.js不仅可用于服务器端，因为其安装完成之后可以用命令行方式方便地调用，因此在个人电脑中也逐渐得到广泛应用。例如为代码编辑器提供插件、用于桌面的Node.js App等。另外还有一个重要的应用领域就是前端自动化，包括代码的预编译/转换（如使用Babel将ECMAScript 6的代码转换为低版本的es代码，将sass/less的样式表文件编译为传统的css文件）、语法检查、代码文件或图像文件的合并、代码的混淆/压缩、自动分发、自动测试等，还可以监视开发文件夹，在内容改变时自动执行上述操作，并自动刷新浏览器页面。这样使得前端领域的开发方式得到了大大进化。

js虽然因为有着一些先天不足而被人诟病，但这些年来却越发展越壮大。这不仅仅是因为依托于浏览器这个宿主环境，更是因为其自身具备的一些优秀特性，Node.js的出现与发展就是一个很好的例证。

**总结**

ajax与Node.js都使用了js的异步回调特性。  
jQuery的出现解决了那个各方面标准尚未统一的混乱时代的许多问题，让js的应用更加广泛，并为未来某些标准的制定指明了方向。  
JSON从js中脱胎而出，作为一种简洁、扩展性好的轻量级数据表示方法，在很多领域得到了广泛使用。  
Node.js在服务器端与开发流程中都越来越得到重视。  
由于Node.js不包含BOM与DOM，因此jQuery不能直接在Node.js上使用，但可以借助jsdom、cheerio之类的库，在构造出虚拟的dom结构后再使用。查看github上的jQuery开发包，可以看到它使用了Node.js上的grunt来进行自动化构建、测试的工作。  
以上几项技术的共同进步，配合浏览器的进步，此外还有硬件条件的发展，让复杂的页面应用越来越多，许多以前在服务器端进行的工作可以转到客户的浏览器中进行，顺应了分布式处理的潮流。

[@于江水](//www.zhihu.com/people/69ed13f3ddc662d6ecd8261512117e85)

**的答案存在一些错误或者不严谨之处：**

1. js设计之初并不是用来弹窗或者改变页面上内容，这些功能是随着浏览器的发展才出现的。实际上js语言自身并不能实现这样的需求。

完整的JavaScript包括ECMAScript、BOM（浏览器对象模型，如window、location等对象）、DOM。其中BOM和DOM虽然是js的组成部分，但js只规定了这两者的接口规范，即浏览器返回这两类对象时，可以用js的方式来处理（点号或者方括号来操作对象成员，可以读取或修改属性等）。也就是说，js脚本通过浏览器提供的接口去操作BOM和DOM，js在其中主要进行流程控制。没有这些接口的支持，js自身是无法完成一些功能的。而早期各家浏览器对于接口实现的不同，也带来了比较严重的兼容性问题。

从js的标准化工作也能看出之间的不同。ECMAScript是由ECMA来进行标准化工作，而BOM和DOM则是由W3C（World Wide Web Consortium）来进行标准化的。

2. jQuery设计的第一要素是对于节点的查询。解决兼容性问题也是jQuery设计的重点，但不是第一位的。当时在尝试解决兼容性问题的js库有好几个，jQuery在这方面并不唯一。而jQuery选择器才是jQuery划时代的特性，从早已存在的事物（css选择器）中发现了价值，并将其修改扩充、发扬光大。

不可否认，jQuery的流行很大程度上要归功于解决兼容性问题，但作为名称中包含的“query”连提都不提，这答案有点偏了。

3. ajax创建之初是用于处理Gmail、Google Maps这样的复杂页面应用，表单提交方式的改变只是ajax技术发展的一个副产品而已。

并且表单提交的问题并不仅仅在于填错项目，实际面临的问题要复杂得多，包括用户的网络掉线、服务器端因为请求过多而暂时无法响应等不可抗因素。在传统模式下，提交表单是前往一个新的动态页，如果出现了错误，运气好的时候点浏览器的“后退”按钮还可以返回表单填写界面，刚才写的东西都还在；运气不好的时候，只会看见一个空白的表单。当时在一些论坛里，总能看到“辛辛苦苦码的字一提交就全没了”的哭诉。

4. 对于JSON的用途描述有错。原答案中提到提交内容简单的时候，用字符串发送，后来改为XML与JSON。实际上，JSON用于客户端（包括浏览器）与服务端通讯的时候，主要指的是服务器端返回处理结果时送回给客户端的数据格式是JSON。而客户端提交给服务器端的数据大多数并不是XML或JSON格式的。

客户端向服务器端提交数据的时候，使用GET或POST方法，其数据一般是简单的“键/值对”表示，例如常见的GET方法在url中构造的参数：

?page=1&pagesize=20

POST方法在提交一般类型的表单时，与GET方法在数据组织形式上基本相近（除非是用multipart类型的表单发送文件数据），但http数据包格式、浏览器历史记录、以及服务器端接收处理有所区别，这属于http协议的内容，此处不展开。  
因此，客户端向服务器端提交数据的时候，主要是采用字符串拼接的方式按上述例子构造，一般不会使用XML或者JSON格式，特别是在ajax出现的早期。（近年来随着页面应用越来越复杂，用JSON格式向服务器端传递数据的情况也逐渐增加）  
而服务器端返回给客户端的数据格式，一开始的主流就是XML，不存在以字符串为主然后过渡到XML的过程。  
另外，XMLHTTP还可以用来进行服务器间通讯，早期发送与接收数据的主流格式都是XML，但是这与ajax不是一回事。

[编辑于 2016-08-26](/question/31305968/answer/116439739)

​赞同 664​​31 条评论

​分享

​收藏​感谢收起​

[于江水](//www.zhihu.com/people/yujiangshui)

[于江水](//www.zhihu.com/people/yujiangshui)

Web 独立开发者

731 人赞同了该回答

===以下回答考虑到简单易懂，更多从使用角度来切入描述，并非从历史和严格的定义切入，严谨的答案参见上面===

有一天，你们人类不满足网页只是一些文字和图片的展示，希望页面上可以有更多功能。比如点击一个按钮，弹一个窗口或者改变页面上某些内容。

为了实现这种功能，就创造了一门脚本语言，逐步升级演化成了 **JavaScript** 这门语言。

**JavaScript 为页面提供更多功能，是页面交互功能的基础语言。此外它的语言规范和引擎还被用于其他领域，比如 Node 等。**

人类为了让自己页面功能更加丰富，使用了大量的 JavaScript，并且写了非常多的代码。这时候发现不同浏览器对 JavaScript 的支持程度非常不统一，而且原生 JavaScript 实现某些看起来很简单的功能都很麻烦。于是 **jQuery** 就把这些兼容性问题统一，并封装了大量的 API，可以让你非常简单就实现很多功能。

**jQuery 屏蔽了浏览器之间的兼容性问题，针对常用功能封装了大量的 API，并支持插件机制，让你写 JS 的效率很高，质量很好。**

简单的页面上的交互再次不满足人类的需求，比如一个资料填写表单，需要填写一个名称，这个名称还必须不能跟之前的重复。校验这个名称不能重复，就需要把数据提交上去，与服务器端数据交互的方法就是 form 提交表单。这时候需要用户填完所有的表单，点击『确定』之后提交校验。如果此时名称被占用就悲剧了。

解决这个问题的方案关键点在数据交互上面。最好的解决方案应该是输入完名字之后，就自动去吧数据发给后端，然后拿到结果并提示给用户，而不是统一提交。于是人类就想出了一套新的数据交互方案，即无刷新的异步请求，名字叫 **Ajax**。通过 Ajax 可以通过 JS 与后端接口进行数据交互，而不会影响当前页面。当接口返回『被占用』的时候，JS 在页面上给个提示就可以很好的实现了。

**Ajax 技术提供了一种新的前后端数据交互方式，不需要刷新页面，而且不阻塞页面执行流程，异步的去请求去获取、交互数据。**

一开始只是通过 Ajax 异步发一个请求，数据就是一个简单的用户名称，所以直接按照字符串发过去就好了。后来需要 Ajax 发送的数据越来越多，比如一整张表单。这时候就需要有一套规则来描述更复杂的数据，一开始估计就是用一些字符分割拼起来，再后来人类用 xml 来描述，发现解析还是挺麻烦的，于是就基于 JavaScript 的数据类型创造了 **JSON** 这种数据描述格式，很简单的就可以描述很复杂的数据。同时独立于语言，这样就可以在多种语言内使用。

**JSON 用来描述前后端数据交互的内容格式，有了 JSON 这样的一套统一的描述规则，前后端解析数据的成本变低，使用非常简单。JSON 属于 JavaScript 的一个子集。**