CSS

1. position取值 及 各自的作用
   1. Static

默认值。没有定位，元素出现在正常的流中（忽略 top, bottom, left, right, z-index）

* 1. Relative

生成相对定位的元素，相对于其正常位置进行定位。

因此，"left:20" 会向元素的 LEFT 位置添加 20 像素。

* 1. Absolute

生成绝对定位的元素，相对于 static 定位以外的第一个父元素进行定位。

元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。

* 1. Fixed

生成绝对定位的元素，相对于浏览器窗口进行定位。

元素的位置通过 "left", "top", "right" 以及 "bottom" 属性进行规定。

* 1. Inherit

规定应该从父元素继承 position 属性的值

1. 水平居中 垂直居中

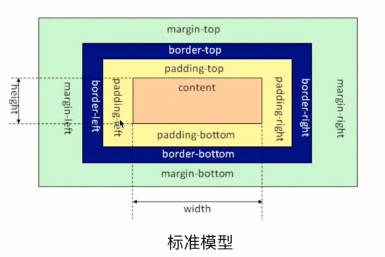
<https://blog.csdn.net/weixin_37580235/article/details/82317240>

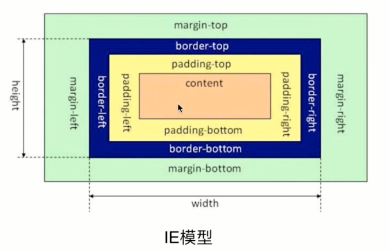
Examples:

[file:///Users/shuxiaoyi/Desktop/align\_center.html](/Users/shuxiaoyi/Documents\\x/Users/shuxiaoyi/Desktop/align_center.html)

1. 盒模型

由里向外content,padding,border,margin.





CSS3 : box-sizing

box-sizing:content-box; /\* 标准模型 \*/

box-sizing:border-box; /\*IE模型\*/

js获取width height

dom.style.width/height /\* style里设置的 \*/

dom.currentStyle.width/height /\* 真正的，only IE \*/

window.getComputedStyle(dom).width/height /\* 同上，兼容性相对较好 \*/

dom.offsetWidth/offsetHeight /\* 兼容性最好 \*/

$(..).height(); // content height

offsetHeight = border + padding + content

clientHeight = content 的可视高度

scrollHeight = 整个元素的高度 包括滚动条和隐藏的部分

1. Flex

<https://www.runoob.com/w3cnote/flex-grammar.html>

flex-direction: row | row-reverse | column | column-reverse;

flex-wrap: nowrap | wrap | wrap-reverse;

flex-flow: <flex-direction> <flex-wrap>;

// 主轴上的对齐方式 justify-content

justify-content: flex-start | flex-end | center | space-between | space-around;

// 交叉轴上的对齐方式

align-items: flex-start | flex-end | center | baseline | stretch;

flex-item属性

order：排列顺序

flex-frow：项目的放大比例，默认为0，即如果存在剩余空间，也不放大；

fled-shrink：项目的缩小比例，默认为1，即如果空间不足，该项目将缩小

flex-basis：分配多余空间之前，项目占据的主轴空间

flex-basis: <length> | auto; /\* default auto \*/

// 允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性

align-self: auto | flex-start | flex-end | center | baseline | stretch;

1. Z-index

<https://www.cnblogs.com/starof/p/4424926.html>

Z-index 仅能在定位元素上奏效（position:relative/absolute/fixed）

stacking context:“堆叠上下文”。每个元素仅属于一个堆叠上下文，z-index描述元素在相同堆叠上下文中“z轴”的呈现顺序

当页面新生成一个box时，它默认的z-index值为auto，意味着该box不会自己产生一个新的local stacking context，而是处于和父box相同的堆叠上下文中

z-index值为0也会生成一个local stacking context，这样该box父box的z-index就不会和其子box做比较，相当于隔离了父box的z-index和子box的z-index

**子元素的z-index值只在父元素范围内有效。相邻的堆叠上下文完全没关系**

不使用z-index的情况时，堆叠顺序（从下到上）：

1. 根元素

2. 正常流中非定位后代元素（按照在HTML中出现的顺序）

3. 已定位后代元素（按照在HTML中出现的顺序）

浮动元素z-index位置介于非定位元素和定位元素之间。（从下到上）

1. 根元素

2. 正常流中非定位后代元素（按照在HTML中出现的顺序）

3. 浮动元素(浮动元素之间是不会出现z-index重叠的)

4. 正常流中inline后代元素

5. 已定位后代元素（这些元素顺序按照HTML文档出现顺序/)

<http://localhost:8080/test/zIndex.html>

1. display: none VS visibility: hidden VS html5 hidden

display:none是彻底消失，不在文档流中占位，浏览器也不会解析；

visibility:hidden是视觉上消失了，类似opacity 0，在文档流中占位，浏览器会解析

visibility:hidden性能上要好，因为不会产生回流。

1. 伪类 伪元素

锚伪类

a:link {color: #FF0000} /\* 未访问的链接 \*/

a:visited {color: #00FF00} /\* 已访问的链接 \*/

a:hover {color: #FF00FF} /\* 鼠标移动到链接上 \*/

a:active {color: #0000FF} /\* 选定的链接 \*/

p:first-child 选择器匹配属于任意元素的第一个子元素的 <p> 元素

p:first-of-type 选择的每个 p 元素是其父元素的第一个 p 元素

:last-child

:last-of-type

:nth-child

:optional

:required

:read-only

:beofre

:after

:first-line

:first-letter

CSS3

1. border-radius & box-shadow & border-image
2. text-shadow & word-wrap
3. @font-face
4. Transform
   1. 2D
      1. translate() 位移
      2. rotate() 旋转
      3. scale() 缩放
      4. skew() 带拉扯的旋转（rotate + scale？）
      5. matrix() 以上所有
   2. 3D
      1. rotateX() 围绕其 X 轴以给定的度数进行旋转
      2. rotateY() 围绕其 Y 轴以给定的度数进行旋转
5. Transition

从一种样式逐渐改变为另一种的效果. 必须规定两项内容：

(1). 规定您希望把效果添加到哪个 CSS 属性上

(2). 规定效果的时长

如需向多个样式添加过渡效果，请添加多个属性，由逗号隔开:

div {

transition: **width** 2s, **height** 2s, **transform** 2s;

}

1. Animation

@keyframes myfirst {

from {background: red;}

to {background: yellow;}

}

div {

animation: myfirst 5s;

}

1. resize
2. box-sizing
3. border-box | content-box
4. outline-offset

超出边框边缘的位置绘制轮廓

div {

border:2px solid black;

outline:2px solid red;

outline-offset:15px;

}

1. CSS3新单位

vw | vh | vmin | vmax

<https://www.cnblogs.com/gopark/p/9173618.html>

LESS

<http://lesscss.cn/>

Less 是一门 CSS 预处理语言，它扩展了 CSS 语言，增加了变量、Mixin、函数等特性，使 CSS 更易维护和扩展。

Less 可以运行在 Node 或浏览器端

Variable

Function

Mixins

Nested Rules

Need to start Tomcat

<http://localhost:8080/test/>

HTML & HTML5

html5新特性

* 1. video
  2. audio
  3. drag & drop

function drag(ev) {

ev.dataTransfer.setData("Text",ev.target.id);

}

function drop(ev) {

ev.preventDefault();

var data=ev.dataTransfer.getData("Text");

ev.target.appendChild(document.getElementById(data));

}

draggable="true"

setData()

ondragstart

ondragover

ondrop

* 1. Canvas

<script type="text/javascript">

var c=document.getElementById("myCanvas");

var cxt=c.**getContext**("2d");

cxt.fillStyle="#FF0000";

cxt.fillRect(0,0,150,75);

</script>

* 1. SVG

指可伸缩矢量图形 (Scalable Vector Graphics)

使用 XML 格式

SVG 图像可通过文本编辑器来创建和修改

SVG 图像可被搜索、索引、脚本化或压缩

SVG 是可伸缩的

SVG 图像可在任何的分辨率下被高质量地打印

SVG 可在图像质量不下降的情况下被放大

* 1. Canvas vs SVG

**Canvas**

依赖分辨率

不支持事件处理器

弱的文本渲染能力

能够以 .png 或 .jpg 格式保存结果图像

最适合图像密集型的游戏，其中的许多对象会被频繁重绘

**SVG**

不依赖分辨率

支持事件处理器

最适合带有大型渲染区域的应用程序（比如谷歌地图）

复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）

不适合游戏应用

* 1. Web 存储

（这块内容后面还有更具体的拓展）

在客户端存储数据

* + 1. localStorage - 没有时间限制的数据存储
    2. sessionStorage - 针对一个 session 的数据存储

当用户关闭浏览器窗口后，数据会被删除

* 1. 应用程序缓存

web 应用可进行缓存，并可在没有因特网连接时进行访问。

三个优势：

* + 1. 离线浏览 - 用户可在应用离线时使用它们
    2. 速度 - 已缓存资源加载得更快
    3. 减少服务器负载 - 浏览器将只从服务器下载更新过或更改过的资源

原生Javascript

1. javascript事件流

<https://www.cnblogs.com/Chen-XiaoJun/p/6210987.html>

<https://www.cnblogs.com/st-leslie/p/5907556.html>

三个阶段：

1. 事件捕获阶段

首先window会获捕获到事件，之后document、documentElement、body会捕获到，再之后就是在body中DOM元素一层一层的捕获到事件

1. 目标阶段
2. 事件冒泡阶段

会和捕获阶段相反的步骤将事件一步一步的冒泡到window

[file:///Users/shuxiaoyi/Desktop/eventFlow.html](/Users/shuxiaoyi/Documents\\x/Users/shuxiaoyi/Desktop/eventFlow.html)

Element.addEventListener 根据第三个参数决定在捕获阶段 或 冒泡阶段触发

Element.onclick 只在冒泡阶段触发；新绑定的事件处理函数会**覆盖**旧的事件处理函数

Element.attachEvent ( IE only )

阻止事件传播

stopPropagation（非IE） 或 cancelBubble（IE）

Event.target: 真正触发事件的dom

Event.currentTarget：通过事件捕获或冒泡 被动触发事件的dom

事件委托：

在JavaScript中，添加到页面上的事件处理程序数量将直接关系到页面的整体运行性能。导致这一问题的原因是多方面的。首先，每个函数都是对象，都会**占用内存**；内存中的对象越多，性能就越差。其次，必须事先指定所有事件处理程序而导致的**DOM访问次数**，会延迟整个页面的交互就绪时间

1. 事件的浏览器兼容性

指的是绑定事件的函数

第一组：

document.getElementById("btn").onclick = function( ){ .... };

document.getElementById("btn").onclick = null;

第二组：（不支持IE）

addEventListener(“click”, function(){...}, false)

removeEventListener())

第三组：（IE 只支持冒泡）

attachEvent(“onclick”, function(){ ... })

detachEvent()

如何支持跨浏览器绑定事件？自己封装一个类

1. Javascript deep copy

<https://www.cnblogs.com/echolun/p/7889848.html>

浅拷贝：

将原对象或原数组的引用直接赋给新对象，新数组/新对象只是原对象的一个引用

深拷贝：

创建一个新的对象和数组，将原对象的各项属性的“值”（数组的所有元素）拷贝过来，是“值”而不是“引用”

Solution1. JSON.parse & JSON.stringify

缺点：

会忽略 undefined

不能处理函数，会被忽略

不能解决循环引用的对象

Solution2. jQuery.extend method

将一个或多个对象的内容合并到目标对象

$.extend( [deep], target, object1 [, objectN ] )

$.extend(object1, object2); /\* object2 合并到 object1 中 \*/

Solution3. 递归

<https://www.imooc.com/article/39933>  
Solution4.ES6 Object.assign(target, source1[, source2, ...]) 只能做到浅拷贝

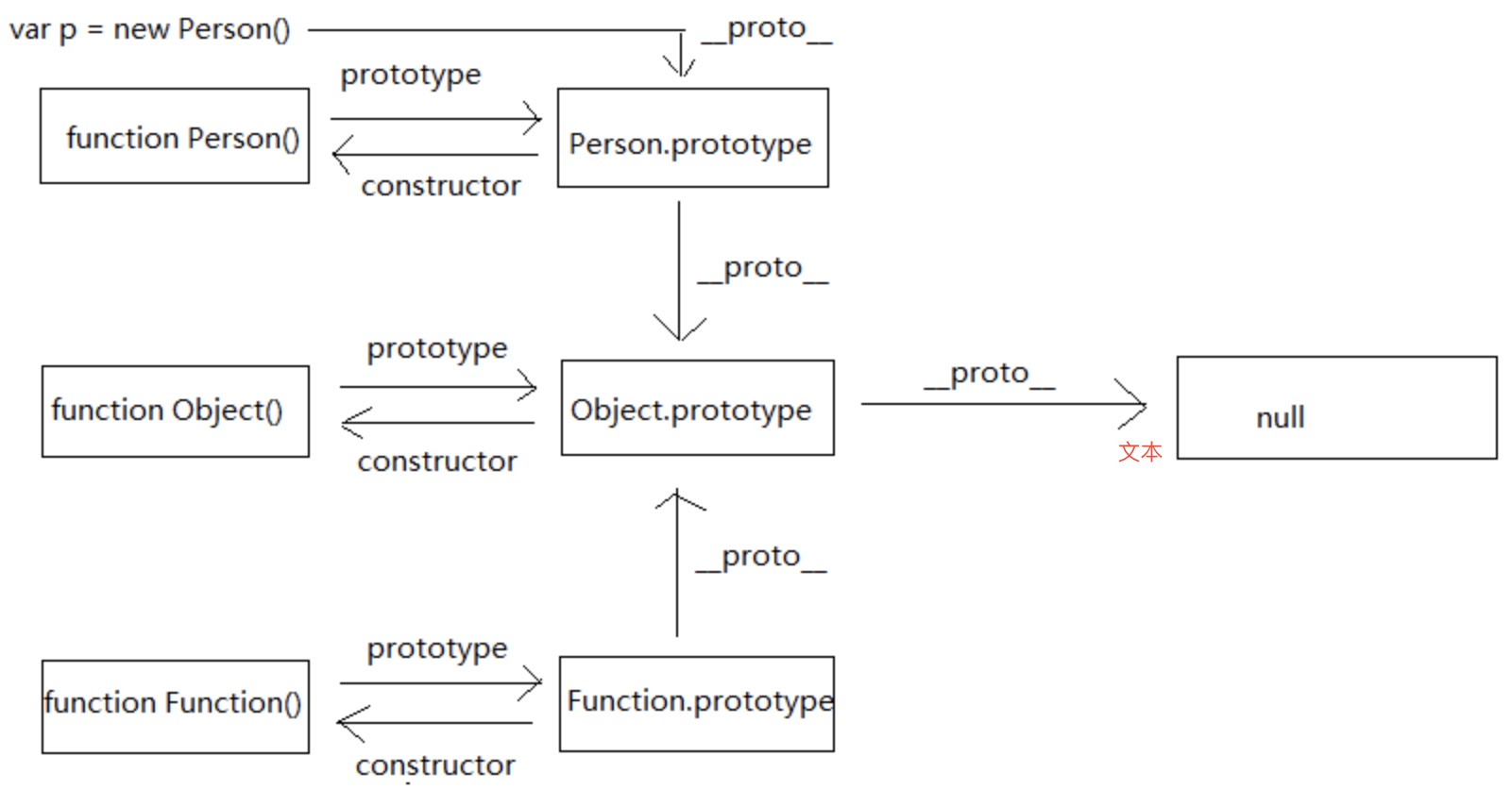
Object.assign 方法只会拷贝源对象自身的并且可枚举的属性到目标对象

不会跳过值为 null 或 undefined 的源对象

1. 原型链 prototype

[file:///Users/shuxiaoyi/Desktop/prortotype\_link.html](/Users/shuxiaoyi/Documents\\x/Users/shuxiaoyi/Desktop/prortotype_link.html)

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Inheritance_and_the_prototype_chain>



结合自己的图，自己的图更完整

所有引用类型（函数，数组，对象）都拥有\_\_proto\_\_属性（隐式原型）

所有函数拥有prototype属性（显式原型）（仅限函数）

原型对象：拥有prototype属性的对象，在定义函数时就被创建

1. String.prototype.replace 两个参数
   1. 第二个参数能否是function？怎么使用？

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace>

returns a **new string** with some or all matches of a pattern replaced by a replacement. The **pattern** can be a **string** or a **RegExp**, and the replacement can be a **string** or a **function** to be called for each match

You can specify a function as the second parameter. In this case, the function will be invoked after the match has been performed. The function's result (return value) will be used as the replacement string. (Note: the above-mentioned special replacement patterns do not apply in this case.) Note that the function will be invoked multiple times for each full match to be replaced if the regular expression in the first parameter is global.

var newStr = str.replace( regexp | substr, newSubstr | function );

var re = /(\w+)\s(\w+)/;

var str = 'John Smith';

var newstr = str.replace(re, '$2, $1');

function replacer(match, p1, p2, p3, offset, string) {

// p1 is nondigits, p2 digits, and p3 non-alphanumerics

return [p1, p2, p3].join(' - ');

}

var newString = 'abc12345#$\*%'.replace(/([^\d]\*)(\d\*)([^\w]\*)/, replacer);

console.log(newString); // abc - 12345 - #$\*%

1. 浏览器存储

<https://www.cnblogs.com/monsterooo/p/6815392.html>

* 1. Cookie
  2. localStorage （H5）
  3. sessionStorage (H5)

document.cookie

window.localStorage; window.sessionStorage

cookie在浏览器请求中每次都会附加到请求头中并发送给服务器。大小：最少要到达4K

localStorage保存数据会一直保存没有过期时间，不会随浏览器发送给服务器。大小5M或更大

sessionStorage仅当前页面有效一旦关闭就会被释放。也不会随浏览器发送给服务器。大小5M或更大

cookie完整格式是：document.cookie=

"key=value[;expires=date][;domain=domain][;path=path][;secure]"

localStorage: 同域的其他页面也能访问

sessionStorage：刷新页面不会清除，但是打开同域新页面访问不到

Summary：

1. cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递。而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。
2. 存储大小限制不同，cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。
3. 数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。
4. 作用域不同，sessionStorage不在不同的浏览器页面中共享，即使是同一个域；localStorage 在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。
5. Web Storage 支持事件通知机制，可以将数据更新的通知发送给监听者。
6. Web Storage 的 api 接口使用更方便，cookie的原生接口不友好，需要自己封装。

安全性

不是什么数据都适合放在 Cookie、localStorage 和 sessionStorage 中的，因为它们保存在本地容易被篡改

1. 浏览器打开网页的过程

<https://www.jianshu.com/p/952142c57813>

<https://www.cnblogs.com/larennani/p/6741289.html> better

自上而下，一边解析，一边渲染

解析html以构建dom树 -> 构建render树 -> 布局render树 -> 绘制render树

1. document.ready、window.onload、body.onload
   1. document.ready: DOM tree构建完成的时候就触发
   2. window.onload: 所有资源都加载完成才触发（各种外部文件，图片，音频，视频）

When bind in js: window.onload = function(){ ... }

When bind in html: <body onload=”...”>...</body>

1. 垃圾回收

<https://www.cnblogs.com/dolphinX/p/3348468.html>

1. 强制类型转换

Javascript深层运用

1. Promise概念和作用

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise>

Code example:

var promise1 = new Promise(function(resolve, reject) {

setTimeout(function() {

resolve('foo');

}, 300);

});

promise1.then(function(value) {

console.log(value); // expected output: "foo"

});

1. Ajax 和 Restful API

<https://blog.csdn.net/m0_37166734/article/details/88604659>

是一种代码风格：URL定位资源，用request method 描述操作

好处：

1. 让url更简洁
2. 提高安全性，减少了parameter 的暴露
3. 容易实现缓存
4. 做到前后端分离。只要规定好请求和返回数据的格式，双方就可以同时开发。后端不用管前端做的页面是否美观，而前端只要考虑如何渲染规定格式的数据。
5. $.ajax 底层实现

<https://www.imooc.com/article/35241>

<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

异步请求数据的web开发技术，对于改善用户的体验和页面性能很有帮助。简单地说，在不需要重新刷新页面的情况下，Ajax 通过异步请求加载后台数据，并在网页上呈现出来。

目的：提高用户体验，较少网络数据的传输量

原理：1. 浏览器让XMLHttpRequest去跟服务器拿点数据

2. 浏览器接着去干别的事情

3. XMLHttpRequest跟服务器请求数据

4. 服务器返回数据给XMLHttpRequest

5. XMLHttpRequest通知浏览器数据来了

6. 浏览器用接收到的数据渲染页面

代码例子：<http://localhost:8080/test/> (need Tomcat)

1. $.deferred

<http://www.cnblogs.com/tiancai/p/5817996.html>

deferred对象就是jQuery的回调函数解决方案

1. $.Deferred() 生成一个deferred对象。
2. deferred.done() 指定操作成功时的回调函数
3. deferred.fail() 指定操作失败时的回调函数
4. deferred.promise() 没有参数时，返回一个新的deferred对象，该对象的运行状态无法被改变；接受参数时，作用为在参数对象上部署deferred接口。
5. deferred.resolve() 手动改变deferred对象的运行状态为"已完成"，从而立即触发done()方法。
6. deferred.reject() 这个方法与deferred.resolve()正好相反，调用后将deferred对象的运行状态变为"已失败"，从而立即触发fail()方法。
7. $.when() 为多个操作指定回调函数。
8. deferred.then()可以把done()和fail()合在一起写，这就是then()方法。

$.when($.ajax( "/main.php" )).then(successFunc, failureFunc );

1. deferred.always()

不管调用的是deferred.resolve()还是deferred.reject()，最后总是执行。

　　$.ajax( "test.html" ).always( function() { alert("已执行！");} );

1. $.when 实现原理

在多个延迟的情况下，如果延迟对象之一被拒绝（rejected），jQuery.when()触发立即 调用 "宿主" 延迟对象的 failCallbacks 回调函数

应用：5个requests，如何保证所有请求都完成后，再进行一些操作？注：即使有某个或某些请求失败了，不可直接结束其他的请求

用计数的方式肯定可以实现，就是不要用$.when

1. 如何优化Performance
   1. 减少requests个数
   2. font-awesome
   3. 压缩javascript
   4. 减少事件的绑定/事件委托
   5. 合理使用localStorage
   6. 合理摆放外部css/js文件的位置
   7. 优化css选择器的写法
2. HTTP请求有哪些
   1. GET / POST / PUT / DELETE / OPTION / ...
   2. GET vs POST

1. GET提交的数据会放在URL之后，以**?**分割URL和传输数据，参数之间以**&**相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的**Body**中.

2. GET提交的数据**大小有限制**（因为浏览器对**URL的长度**有限制），而POST方法提交的数据**没有限制**.

3. GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。

4. **GET方式提交数据，会带来安全问题**，比如一个登录页面，通过GET方式提交数据时，用户名和密码将**出现在URL**上，如果页面可以被缓存或者其他人可以访问这台机器，就可以**从历史记录获得该用户的账号和密码**.

* 1. POST vs PUT

新建一条记录的话就用post，

更新一条记录的话就用put.

POST是非幂等的

PUT是幂等的

幂等：传入的参数相同，返回的结果一定也相同

* 1. PATCH

局部更新（你眼里的增量）

1. HTTP协议有哪些

http, https, ftp

<https://www.cnblogs.com/rayray/p/3729533.html>

1. HTTP response status 有哪些
   1. 1XX 提示信息 - 表示请求已被成功接收，继续处理
   2. 2XX 成功 - 表示请求已被成功接收，理解，接受
   3. 3XX 重定向 - 要完成请求必须进行更进一步的处理
   4. 4XX 客户端错误 - 请求有语法错误或请求无法实现
   5. 5XX 服务器端错误 - 服务器未能实现合法的请求

200：响应成功

301：永久重定向/永久转移

302：临时重定向/临时转移

304：本次获取内容是读取缓存中的数据

400：请求参数错误

401：无权限访问

404：访问的资源不存在

1. 跨域概念

<https://www.cnblogs.com/yongshaoye/p/7423881.html>

<https://segmentfault.com/a/1190000015597029> better

* 1. 概念

跨域是指从一个域名的网页去请求另一个域名的资源

浏览器的同源策略（用来防御来自邪门歪道的攻击）

只要 协议，域名，端口有任何一个的不同，就被当作是跨域

* 1. 为什么浏览器要限制跨域访问

安全问题：如果一个网页可以随意地访问另外一个网站的资源，那么就有可能在客户完全不知情的情况下出现安全问题

* 1. 为什么要跨域

有时公司内部有多个不同的子域

1. 跨域解决方案
   1. JSONP

在HTML标签里，一些标签比如script、img这样的获取资源的标签是没有跨域限制的

* 1. 空iframe加form
  2. CORS
  3. 代理
  4. postMessage
  5. document.domain 只适用于不同子域的框架间的交互
  6. window.name

1. 安全漏洞 + 如何预防

Cookie

GET请求

1. Chrome 运行时性能瓶颈分析

<https://juejin.im/post/5cd15712e51d453a393af4c5>

ES6

1. Babel

一个广泛使用的 ES6 转码器，可以将 ES6 代码转为 ES5 代码，从而在现有环境执行。这意味着，不用担心兼容性.Babel 默认只转换新的 JavaScript 句法（syntax），而不转换新的 API

1. let 命令

只在let命令所在的代码块内有效

没有声明提前

暂时性死区：只要块级作用域内存在let命令，它所声明的变量就“绑定”这个区域，不再受外部的影响（不许往外看！！）ES6 明确规定，如果区块中存在let和const命令，这个区块对这些命令声明的变量，从一开始就形成了封闭作用域。凡是在声明之前就使用这些变量，就会报错

不允许在相同作用域内，重复声明同一个变量

1. 块级作用域

内层变量可能会覆盖外层变量（可以与声明提前相关）

用来计数的循环变量泄露为全局变量

内层作用域可以定义外层作用域的同名变量

立即执行函数表达式不再必要

避免在块级作用域内声明函数

必须有大括号

1. const命令

声明一个只读的常量。一旦声明就不能改变

一旦声明变量，就必须立即初始化

块级作用域 √

没有声明提前 √

暂时性死区 √

不能重复声明 √

实际上保证的并不是变量的值不得改动，而是变量指向的那个**内存地址**所保存的数据不得改动。所以要小心非简单数据类型

var命令和function命令声明的全局变量，依旧是顶层对象的属性；另一方面规定，let命令、const命令、class命令声明的全局变量，不属于顶层对象的属性

1. 解构赋值

按照一定模式，从数组和对象中提取值，对变量进行赋值

只要等号两边的模式相同，左边的变量就会被赋予对应的值

eg. let [a, b, c] = [1, 2, 3];

AngularJS

- 双向绑定原理

- 工作机制

- 生命周期

Vue

- 双向绑定原理

- 工作机制

- 生命周期

- VUX (mobile)

- Element UI

- 为什么要用$emit 而不允许子component修改父component的属性值

有这样一个例子：在父组件中有一个列表，双击其中一个元素进行编辑，该元素的数据作为props传递给一个子组件，在子组件中需要对该数据进行编辑，你会发现如上所说，编辑后父组件的值也发生了变化。

<https://www.cnblogs.com/leaf930814/p/6891254.html>

<https://segmentfault.com/a/1190000006599500>

KnockouJS

- 双向绑定原理

- 工作机制

- 生命周期

Webpack

<https://www.webpackjs.com/>

JavaScript 应用程序的静态模块打包器(module bundler)

递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个 bundle

Nodejs

设计模式 及 js中的具体运用

<https://addyosmani.com/resources/essentialjsdesignpatterns/book/>