**HTML+CSS**

1. **区分input和textarea的区别**

<input>是一个单行输入框，有value属性（value属性指定初始值），但是它不能自动换行；用来放置字数较少的单行文字内容

<textarea>是一个多行输入框，没有value属性，但是它能自动换行；一般让用户可以输入多行文字,输入的文字信息量相比较大

1. **用一个div模拟textarea的实现**

contenteditable="true"

1. **左右布局：左边定宽、右边自适应，不少于3种方法**

1.使用float

2.使用position 父级必须：relative;左侧position:absolute left:0px;右侧：margin-left:10px;

3.table 不设置position

4.使用flex属性；（flex 属性用于设置或检索弹性盒模型对象的子元素如何分配空间。）

1. **relative和absolute的区别：**

absolute：它只坚持一点，就以**父级左上角**为原点进行定位，父级的padding对其根本没有影响。 (父级元素position 的属性又不能为absolute，不然就会以浏览器左上角为原点了，所以父级元素的position 属性只能为relative！)

relative：默认参照父级的原始点为原始点，无父级则以文本流的顺序在上一个元素的底部为原始点，配合TRBL 进行定位，当父级内有padding 等CSS 属性时，当前级的原始点则参照父级内容区的原始点进行定位

注：任何元素的默认position 的属性值均是static,静态。

如果用position来布局页面，父级元素的position属性必须为relative，而定位于父级内部某个位置的元素，最好用absolute，因为它不受父级元素的padding的属性影响，当然你也可以用relative，不过到时候计算的时候不要忘记padding的值。   
但是定位(position)有一个缺点，不会自适应内部元素的高度，所以平时我们在布局页面的时候，如果某个或者某些模块高度永远不变，就可以用定位，但是建议大家布局页面的时候，还是要以Float 为主，Position 为辅！这样你才能做出高质量的页面。   
最后，请大家记住这句话:“布局页面，Float为主，Position为辅！

1. **padding和margin的区别，同时使用会怎样？**

 margin（上右下左）和padding都是用来隔开元素，margin用于布局元素使元素与元素互不相干，padding是用于元素与内容之间的间隔。

使用margin（外边距）的情景：

1.需要在border外侧添加空白的时候

2.空白处不需要有背景色时

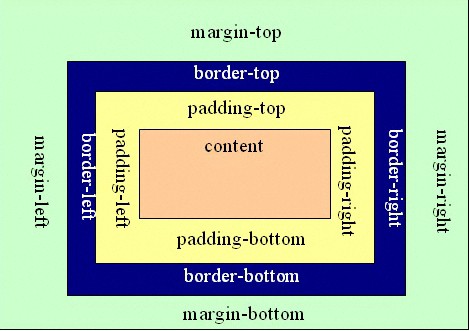
3.上下相连的两个盒子之间的空白需要相互抵消时

使用padding（内边距）的情景：

1.需要再border内侧添加空白时（往往是文字与边框距离的设置）

2.空白处需要背景色时

3.上下相连的两个盒子之间的空白希望等于两者之和时。

s

同时使用就既有外边距的效果也有内边距的效果

1. **CSS的盒子模型以及box-sizing属性**

所有HTML元素可以看作盒子，在CSS中，"box model"这一术语是用来设计和布局时使用。CSS盒模型本质上是一个盒子，封装周围的HTML元素，它包括：边距，边框，填充，和实际内容。

margin（外边距）padding（内边距）border（边框）content（内容（盒子的内容））

最终元素的总宽度计算公式是这样的：

总元素的宽度=宽度+左填充+右填充+左边框+右边框+左边距+右边距

元素的总高度最终计算公式是这样的：

总元素的高度=高度+顶部填充+底部填充+上边框+下边框+上边距+下边距

（1）有两种， IE 盒子模型、W3C 盒子模型；

（2）盒模型：内容(content)、填充(padding)、边界(margin)、边框(border)；

（3）区别： IE的content部分把 border 和 padding计算了进去;

盒子模型的box-sizing：content-box|border-box|inherit

1）content-box:

此值维持css2.1盒模型的组成模式，border|padding|content {element width=border+padding+content}

padding和border不被包含在定义的width和height之内。对象的实际宽度等于设置的width值和border、padding之和，即 ( Element width = width + border + padding )此属性表现为标准模式下的盒模型。

2)border-box:(在解决自适应布局方面有优势)

此值改变css2.1盒模型组成形式，content|border|padding{element width=content}

padding和border被包含在定义的width和height中，对象的实际高度等于设置的width值，即使定义有border和padding也不会改变对象的实际高度，即（Element width = width）

1. **一个div里面包着另一个div，里面的div外边距为15%，怎么排列**

里面的外边距是外面div宽度的15%

1. **CSS同时水平居中和垂直居中**

1.水平居中：margin: 0 auto; text-align:center;

2.水平垂直居中

①已知子div高度宽度用position 父级：relative 子：absolute+left/top50%+margin(-子高/2 0 0 -子宽/2)

②未知子div高度宽度用position 父级：relative 子：absolute+left/top50%+transform：translate(-50%,-50%)

注：transform：none|*transform-functions（旋转div元素）其中translate（x,y）定义2D转换*

③父级：display：flex；

justify-content: center;//使子项目水平居中

align-items: center;//使子项目垂直居中

④table-cell布局（不推荐使用）

注：css3 Flex布局（伸缩布局盒模型）

其中容器的属性用的比较多：

* flex-direction、flex-wrap、flex-flow、justify-content、align-items、align-content

1. **静态布局、自适应布局、流式布局、响应式布局、弹性布局**

1.静态布局：static layout传统Web设计，网页上的所有元素的尺寸一律使用px作为单位。

2.流式布局：fluid layout**是页面元素的宽度按照屏幕分辨率进行适配调整，但整体布局不变。**代表作**栅栏系统**（网格系统）。网页中主要的划分区域的尺寸使用百分数（搭配min-\*、max-\*属性使用），例如，设置网页主体的宽度为80%，min-width为960px。图片也作类似处理（width:100%, max-width一般设定为图片本身的尺寸，防止被拉伸而失真）。

3.自适应布局：adaptive layout 自适应布局的特点是分别为不同的屏幕分辨率定义布局，即创建多个静态布局，每个静态布局对应一个屏幕分辨率范围。改变屏幕分辨率可以切换不同的静态局部（页面元素位置发生改变），但在每个静态布局中，页面元素不随窗口大小的调整发生变化。可以把自适应布局看作是静态布局的一个系列。

4.响应式布局：responsive layout 随着CSS3出现了**媒体查询**技术，又出现了**响应式设计**的概念。响应式设计的目标是确保一个页面在所有终端上（各种尺寸的PC、手机、手表、冰箱的Web浏览器等等）都能显示出令人满意的效果，对CSS编写者而言，在实现上不拘泥于具体手法，但通常是糅合了流式布局+弹性布局，再搭配媒体查询技术使用。——分别为不同的屏幕分辨率定义布局，同时，在每个布局中，应用流式布局的理念，即页面元素宽度随着窗口调整而自动适配。即：创建多个流体式布局，分别对应一个屏幕分辨率范围。可以把响应式布局看作是流式布局和自适应布局设计理念的融合

流式布局是用于解决类似的设备不同分辨率之间的兼容(一般分辨率差异较少)；响应式是用于解决不用设备之间不用分辨率之间的兼容问题(一般是指PC，平板，手机等设备之间较大的分辨率差异)。

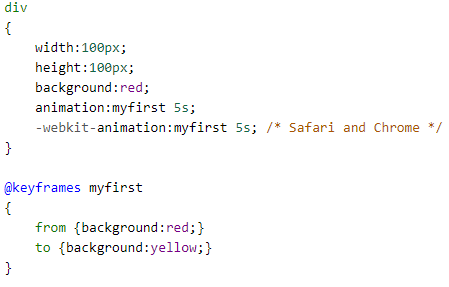
5.弹性布局（rem/em布局）：**rem,em区别：**rem,em都是顺应不同网页字体大小展现而产生的。其中，em是相对其父元素，在实际应用中相对而言会带来很多不便；而rem是始终相对于html大小，即页面根元素。

这类布局的特点是，**包裹文字的各元素的尺寸采用em/rem做单位，而页面的主要划分区域的尺寸仍使用百分数或px做单位（同「流式布局」或「静态/固定布局」）**。**早期浏览器不支持整个页面按比例缩放**，仅支持网页内文字尺寸的放大，这种情况下。使用em/rem做单位，可以使包裹文字的元素随着文字的缩放而缩放

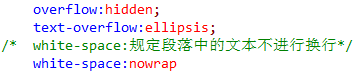
注：浏览器的默认字体高度一般为16px，即1em:16px，但是 1:16 的比例不方便计算，为了使单位em/rem更直观，CSS编写者常常将页面跟节点字体设为62.5%，比如选择用rem控制字体时，先需要设置根节点html的字体大小，因为浏览器默认字体大小16px\*62.5%=10px。。

1. **CSS3的动画用过吗？**

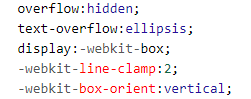
当在@keyframes创建动画时把name（动画名称）和5s(动画时长)绑定给div元素；



1. **单行文本溢出**



**多行文本溢出：**

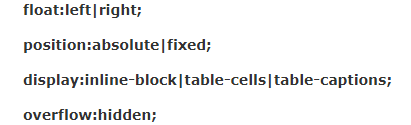
****

1. **BFC布局(尚不完善)**

block formatting contexts(块级格式化上下文)：具有BFC特性的元素可以看做是隔离了的独立容器，容器里面的元素不会在布局上影响到外面的元素；

触发条件：

　　【1】根元素，即HTML元素   
　　【2】float的值不为none   
　　【3】overflow的值不为visible   
　　【4】display的值为inline-block、table-cell、table-caption   
　　【5】position的值为absolute或fixed用代码表示如下：

****

BFC不会与float元素相互覆盖

**BFC和IFC的区别：**

IFC: inline formatting context

1.框会从包含块的顶部开始，一个接一个地水平摆放。

2.摆放这些框的时候，它们在水平方向上的外边距、边框、内边距所占用的空间都会被考虑在内。在垂直方向上，这些框可能会以不同形式来对齐：它们可能会把底部或顶部对齐，也可能把其内部的文本基线对齐。能把在一行上的框都完全包含进去的一个矩形区域，被称为该行的行框。水平的margin、padding、border有效，垂直无效。不能指定宽高。

3. 行框的宽度是由包含块和存在的浮动来决定。行框的高度由行高计算这一章所描述的规则来决定。

1. **opcity visibility display 元素不可见的区别**

opcity : 0 元素隐藏占位有事件（子元素能继承，但是不能通过设置子元素opacity: 0使其重新显示）  
visibility ： hidden 元素隐藏占位无事件（子元素可以继承，可以通过设置子元素visibility:visible使子元素显示出来）  
display ： none 元素隐藏不占位无事件（子元素不能继承，彻底没了）

css最后

**JavaScript**

JavaScript组成：

①ECMAScript（核心European Computer Manufacturers Association）：作为核心，它规定了语言的组成部分：语法、类型、语句、关键字、保留字、操作符、对象

②DOM（document object model）：文档对象模型：DOM把整个页面映射为一个多层节点结果，开发人员可借助DOM提供的API，轻松地删除、添加、替换或修改任何节点。

③BOM（browser object model）:浏览器对象模型：支持可以访问和操作浏览器窗口的浏览器对象模型，开发人员可以控制浏览器显示的页面以外的部分。

## ④ES5作为ECMAScript第五个版本（第四版因为过于复杂废弃了）

1. **JS基本数据类型**

字符串、数字（可以带小数点也可以不带）、布尔（true和false）、数组（new Array()）、对象(name:value)、Null()、Undefined(表示变量不含有值可以通过将变量的值设为null来清空变量)

js拥有动态模型，所以相同的变量可以用作不同类型

js变量均为对象，当你声明一个变量时（new）就创建了一个新的对象；

1. **数字字符串转换方法**

数字--字符串：①toString()②在数字后面加一个空字符串

字符串--数字：①转换函数：parseInt(),parseFloat()②强制类型转换：Number(100)-->100③利用js变量弱类型转换：

varstr= '012.345 ';  
var x = str-0;  
x = x\*1;（不推荐）

1. **JS事件流机制**

js和HTML元素的交互几乎是通过事件来完成的，事件从出发到完成响应一般分为：捕获阶段、目标阶段、冒泡阶段；

**事件流：**从页面中接受事件的顺序；

事件冒泡（IE）：从里向外（所有现代浏览器都支持冒泡事件）（也就是说，click事件首先在div元素上发生，然后click事件**沿DOM树向上传播**，在每一级节点上都会发生，直至传播至document对象。）

Netscape提出了“事件捕获”：从外到里的顺序（事件到达预定目标之前捕获它）

事件捕获阶段：事件从最上一级标签开始往下查找，直到捕获到事件目标(target)。

事件冒泡阶段：事件从事件目标(target)开始，往上冒泡直到页面的最上一级标签。

1. **操作dom动态创建元素**

①Document.write()，然后通过某个元素的appendChild(node);方法将创建的元素添加到相应的元素下。会发生一个问题：默认情况之下，页面由上而下地加载，形成一个文档流，当执行完毕时，文档流就会关闭，从而将之前生成的元素冲刷掉，所以不推荐使用这种方法。

②.innerHTML

③document.createElement()创建元素

1. **JS的event对象**

①在触发事件的函数中会接收到event对象，通过该对象我们需要的一些参数，比如说我们需要知道该事件作用到谁身上了，可以通过target属性获得；

②IE属性；

1. **事件委托（和注册事件有关）**

给父元素绑定事件--监听子元素的冒泡事件--找到是哪个子元素的事件

document.getElementsByTagName("ul")[0].addEventListener('click',function(e){

    alert("点击的内容是："+e.target.innerHTML);

});

事件委托的好处：1. 事件委托技术可以避免对每个子元素添加事件监听器，减少操作**DOM节点**的次数，从而减少浏览器的重绘和重排，提高代码的性能。

2. 使用事件委托，只有父元素与DOM存在交互，其他的操作都是在JS虚拟内存中完成的，这样就大大提高了性能。

1. **JS中的this**

1.全局作用域或者普通函数中的this指向全局对象window；

2.方法调用中谁调用this就指向谁；

3.在构造函数或者构造函数原型对象中 this 指向构造函数的实例

4.箭头函数中指向外层作用域的this

5.非普通函数中的this：

　情况1：如果一个函数中有this，但是它没有被上一级的对象所调用，那么this指向的就是window。

　　情况2：如果一个函数中有this，这个函数有被上一级的对象所调用，那么this指向的就是上一级的对象。

　　情况3：如果一个函数中有this，**这个函数中包含多个对象，尽管这个函数是被最外层的对象所调用，this指向的也只是它上一级的对象，**

情况4：对象中的嵌套函数的this指向不是当前对象，而是window。

解决方法：bind()/用中介接受一下上一层函数的this/箭头函数

1. **this和super的区别**

this指向函数所在的当前对象

super指向当前对象的原型对象

super.name

等同于

Object.getPrototypeOf(this).name【属性】

等同于

Object.getPrototypeOf(this).name.call(this)【方法】

///Object.setPrototypeOf(obj, prototype)

Class中的 super()，它在这里表示父类的构造函数，用来新建父类的 this 对象

super()相当于Parent.prototype.constructor.call(this)

ES5的继承，实质上是先创造子类的实例对象this，然后再将父类的方法添加到this上（Parent.call(this)）.  
ES6的继承，需要先创建父类的this，子类调用super继承父类的this对象，然后再加工。

 super 在静态方法之中指向父类，在普通方法之中指向父类的原型对象；

1. **JS的call、apply、bind**

在javascript中，call 和 apply 都是为了改变某个函数运行时的上下文（context）而存在的，换句话说，就是为了改变函数体内部 this 的指向。

对于 apply、call 二者而言，作用完全一样，只是接受参数的方式不太一样；

apply：最多只能有两个参数——新this对象和一个数组argArray。如果给该方法传递多个参数，则把参数都写进这个数组里面，当然，即使只有一个参数，也要写进数组里。如果argArray不是一个有效的数组或arguments对象，那么将导致一个TypeError。如果没有提供argArray和thisObj任何一个参数，那么Global对象将被用作thisObj，并且无法被传递任何参数。

call：它可以接受多个参数，第一个参数与apply一样，后面则是一串参数列表。这个方法主要用在js对象各方法相互调用的时候，使当前this实例指针保持一致，或者在特殊情况下需要改变this指针。如果没有提供thisObj参数，那么 Global 对象被用作thisObj。

bind() 方法与 apply 和 call 很相似，也是可以改变函数体内 this 的指向。

MDN的解释是：bind()方法会创建一个新函数，称为绑定函数，当调用这个绑定函数时，绑定函数会以创建它时传入 bind()方法的第一个参数作为 this，传入 bind() 方法的第二个以及以后的参数加上绑定函数运行时本身的参数按照顺序作为原函数的参数来调用原函数。

1. **判断一个对象是不是空对象**

1.讲对象转换成字符串：使用JSON.stringify()这个方法对对象进行强转

2.for in 循环

functionisEmptyObject(obj){

for(var key inobj){

returnfalse

};

returntrue

};

3.jQuery的isEmptyObject()方法实现的代码即简单又简洁（实质就是2）

1. JS的RegExp对象

varpatt=new RegExp(pattern,modifiers);

* modifiers(修饰符) 用于指定全局匹配（g）、区分大小写(大小写不敏感i)的匹配和多行匹配（m）

function getUrlPara(name) {正则表达式

varreg = new RegExp("(^|&)" + name + "=([^&]\*)(&|$)","i");

var r = window.location.search.substr(1).match(reg);

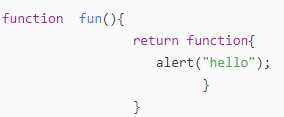
if (r!=null) return (r[2]); return null;

}

1. **POST一般可以发送什么类型的文件**

？？？json+xml+text

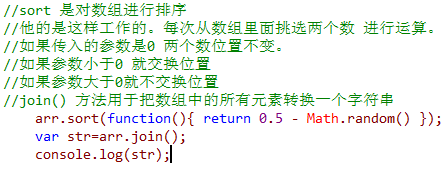
1. **JS的闭包，优劣：**

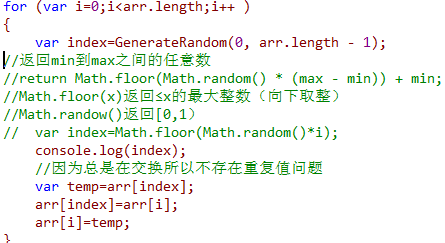
  闭包javascript是最强大的特征之一，它**允许函数访问局部作用域之外的函数**。创建闭包的常见方式：就是在一个函数里创建另一个函数

好处：1.缓存一些耗时的函数对象的值2.面向对象中的对象（虽然JavaScript中没有类这样的机制，但是通过使用闭包，我们可以模拟出这样的机制。）3.匿名自执行函数，执行完后很快就会被释放，关键是这种机制不会污染全局对象。

坏处：1.内存消耗大2.使用闭包时，会涉及到跨作用域访问，每次访问都会导致性能损失。

1. **JS打乱一组排好序的数组**

**或者**

****

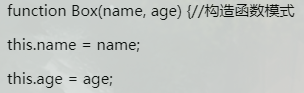
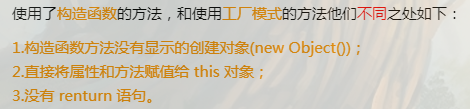
1. **给定长度为N的排好序的数组，要求给出数组元素为M的情况。**

利用for循环遍历

1. **JS的面向对象和原型：**

JS中并没有类的概念；

对象：利用工厂模式解决重复实例化的问题；但对象无法识别

ES中使用构造函数解决对象识别问题：但并没有new Object()

构造函数和普通函数的唯一区别，就是他们调用的方式不同。只不过，构造函数也是函数，必须用 new 运算符来调用，否则就是普通函数。

　构造函数的特点：

　　　　a:构造函数的首字母必须大写，用来区分于普通函数

　　　　b:内部使用的this对象，来指向即将要生成的实例对象

　　　　c:使用New来生成实例对象

构造函数的缺点：

　　　　所有的实例对象都可以继承构造器函数中的属性和方法。但是，同一个构造函数的对象实例之间，无法共享属性和方法。

　　　　解决思路：

　　　　　　a:所有实例都会通过原型链引用到prototype

　　　　　　b:prototype相当于特定类型所有实例都可以访问到的一个公共容器

　　　　　　c:那么我们就将重复的东西放到公共容易就好了

原型：我们创建的每个函数都有一个 prototype(原型)属性，这个属性是一个对象，它的用途是包含可以由特定类型的所有实例共享的属性和方法。逻辑上可以这么理解：prototype 通过调用构造函数而创建的那个对象的原型对象。**使用原型的好处可以让所有对象实例共享它所包含的属性和方法**。也就是说，不必在构造函数中定义对象信息，而是可以直接将这些信息添加到原型中。在原型模式声明中，多了两个属性，这两个属性都是创建对象时自动生成的。\_\_proto\_\_属性是实例指向原型对象的一个指针，它的作用就是指向构造函数的原型属性 constructor。通过这两个属性，就可以访问到原型里的属性和方法了。

1. **JS实现链表：**

在某些编程语言中，数组的长度是预先设定好的，想要额外添加元素或者删除元素是一件比较困难的事。那么使用链表的话恰恰就解决了这些问题，对于链表来说删除或添加一个元素是非常方便的，除了数据的随机访问(可以实现但是比较麻烦，比如可以通过添加和操作索引值来实现)，它几乎可以用在任何可以使用一维数组的情况中。

链表的定义：　链表是由一组节点组成的集合。每个节点都使用一个对象的引用指向它的后继。指向另一个节点的引用叫做链。

1. 算法：

**冒泡和快速排序：**



**快速排序:**

****

1. **0.1+0.2===0.3吗**

不是；实际上是因为计算机内部的信息都是由**二进制方式**表示的，即0和1组成的各种编码，但由于**某些浮点数没办法用二进制准确的表示出来**，也就带来了一系列精度问题。当然这也**不是JS独有的问题**。

1. **JS设计模式**

1.工厂模式

function Parent(){

var Child = new object();

Child.name = "李小龙";

Child.age = "30";

return Child;

};

2.构造函数模式

function Parent(){

this.name = "李小龙";

this.age = "30";

};

3.JS原型模式

function Parent(){

Parent.prototype.name = "李小龙";

Parent.prototype.age = "30";

};

4.构造函数+原型的js混合模式

5.构造函数+原型的动态原型模式

**HTTP协议的作用原理:**

连接: Web浏览器与Web服务器建立连接，打开一个称为socket (套接字) 的虚拟文件，此文件的建立标志着连接建立成功；

请求: Web浏览器通过socket向Web服务器提交请求，HTTP的请求一般是GET或POST命令(POST用于FORM参数的传递)；

GET命令的格式为:GET 路径/文件名HTT文件名指出所访问的文件，HTTP/1.0指出Web浏览器使用的HTTP版本；

应答: Web浏览器提交请求后，通过HTTP协议传送给Web服务器；

Web服务器接到后，进行事务处理，处理结果又通过HTTP传回给Web浏览器，从而在Web浏览器上显示出所请求的页面。

1. **GET和POST的区别**

get把请求的数据放到url上；post放在request body中

get提交数据最大是2k； post理论上是没有限制的

get产生一个tcp数据包；post产生两个tcp数据包

get在浏览器回退时无害；post会重新提交请求

get请求只接受ASCII字符；post没限制

1. **Cookie是什么**

cookie可以认为是一个记录了简单内容的文本文件；它直接**绑定**在HTML页面中；

前端操作：写入：document.cookie=key+'='+value+';expires='+过期时间戳'；

读取（常用）：console.log(document.cookie);

删除：document.cookie=key+'='+value+';expires='+当前时间戳+1ms'；

**session是什么**

session是存放在服务器端类似于hashtable结构来存放用户数据；

作用：实现网页之间数据传递，是一个存储在服务器端的对象集合；

1. **cookie和session的区别**

①cookie放在客户浏览器中；session数据放在服务器上

②cookie不是很安全

③session一定时间内放在服务器上，访问增多比较占用服务器资源，从减轻服务器负担的角度看，应当使用cookie

④单个cookie保存的数据不能超过**4k**；Session对象没有对存储的数据量的限制，其中可以保存更为复杂的数据类型；

两者最大的区别在于生存周期，一个是IE启动到IE关闭.(浏览器页面一关 ,session就消失了)一个是预先设置的生存周期，或永久的保存于本地的文件。(cookie)

**联系：**

session是通过cookie工作的；session和cookie之间通过$\_COOKIE['PHPSESSID']联系，通过其可以知道session的id

在[购物网站](http://www.so.com/s?q=%E8%B4%AD%E7%89%A9%E7%BD%91%E7%AB%99&ie=utf-8&src=internal_wenda_recommend_textn)中通常将用户加入[购物车](http://www.so.com/s?q=%E8%B4%AD%E7%89%A9%E8%BD%A6&ie=utf-8&src=internal_wenda_recommend_textn)的[商品](http://www.so.com/s?q=%E5%95%86%E5%93%81&ie=utf-8&src=internal_wenda_recommend_textn)联通session\_id[记录](http://www.so.com/s?q=%E8%AE%B0%E5%BD%95&ie=utf-8&src=internal_wenda_recommend_textn)到数据库中，当用户再次访问是，通过sessionid就可以查找到用户上次加入购物车的商品。因为sessionid是唯一的，记录到数据库中就可以根据这个查找了。

1. **localstorage和sessionstorage区别**

web Storage 是HTML 5引入的一个重要的功能，在前端开发的过程中会经常用到，它可以在客户端本地存储数据，类似cookie，但其功能却比cookie强大的多。cookie的大小只有4Kb左右（浏览器不同，大小也不同），而web Storage的大小有5MB。

localstorage：生命周期是永久性的。假若使用localStorage存储数据，即使关闭浏览器，也不会让数据消失，除非主动的去删除数据，使用的方法如上所示。localStorage有length属性，可以查看其有多少条记录的数据。

sessionstorage：生命周期是在浏览器关闭前。也就是说，在整个浏览器未关闭前，其数据一直都是存在的。sessionStorage也有length属性，其基本的判断和使用方法和localStorage的使用是一致的。