目录

1.响应式布局设计

参考《HTML5权威指南》P109-113 http://www.educity.cn/jianzhan/403677.html

(1)什么是响应式布局

一个网站能够兼容多个终端——而不是为每个终端做一个特定的版本。

（2）响应式布局的优点和缺点

优点

- 面对不同分辨率设备灵活性强
- 能够智能地解决多设备展示适应问题

缺点

- 为了兼容各种设备，工作量很大
- 代码累赘，会出现隐藏无用的元素，加载时间加长
- 一证程度上改变了网站原有的布局结构

（3）如何实现

示例

a.在style 标签中使用media属性

```
<style media="screen AND (max-width:500px)" type="text/css">
    a{

    }
    #id{
    }

</style>
<style media="screen AND (min-width:500px)" type="text/css">
</style>
```

b.在样式表中内嵌@media

```
<style type="text/css">
    body{
        width: 100%;

    }
    @media(min-device-width:1024px) and (min-width:900px){
        div{
            position: absolute;
            width: 60%;
            left: 20%;
            height: 20px;
            background-color: yellow;
        }
    }
    @media screen and (max-width:800px){
        div{
            position: absolute;
            width: 98%;
            left: 1%;
            height: 20px;
            background-color: yellow;
        }
    }

</style>
```

c.在link标签中使用media属性

```
    <link rel="stylesheet" type="text/css" media="only screen and （max-width：480px）, only screen and （max-device-width：480px）
```

语法

字符间以空格相连。选取条件包含在小括号内，only（限定某种设备，可省略），and（逻辑与），not（排除某种设备）为逻辑关键字。多种设备用逗号分隔。这一点继承了css基本语法。

2.CSS3新增内容

（1）CSS动画 *优先说*

示例：

```
 .logoWave{
        animation: scaleToBigger 0.5s ease-in-out 1;
        -webkit-animation: scaleToBigger 0.5s ease-in-out 1;
    }
    @-webkit-keyframes scaleToBigger{
        0%{
            -webkit-transform: scale(1.0);
            opacity: 1;
        }
        100%{
            -webkit-transform: scale(1.3);
            opacity: 0;
        }
    }
```

@keyframes规则

通过 @keyframes 规则，能够创建动画。- 目前浏览器都不支持 @keyframes 规则。- Firefox 支持替代的 @-moz-keyframes 规则。- Opera 支持替代的 @-o-keyframes 规则。- Safari 和 Chrome 支持替代的 @-webkit-keyframes 规则。

animation属性

提供该属性，将动画与元素绑定

- animation简写属性语法。

  ```
  animation: name duration timing-function delay iteration-count direction;
  ```

- animation-name 指定动画的名称。即@-webkit-keyframes后面所接名称。

- animation-duration 设置动画持续播放的时间。（<时间>）
- animation-time-function 如何计算中间动画值。（ease，linear,ease-in,ease-out,ease-in-out,cubic-bezier
- animation-delay 设置动画开始前的延迟（<时间>）
- animation-iteration-count 设置动画播放次数（<数值>）
- animation-direction 设置动画循环播放时是否逆序反向播放（normal,alternate）

(2)CSS过渡 *优先说*

实例

```
    #banana{
        font-size: large;
        border: medium solid black;
        background-color: white;
        color: black;
    }
    #banana:hover{
        font-size: xx-large;
        border: medium solid white;
        background-color: green;
        color: white;
        -webkit-transition: background-color,color,font-size,border 1s linear .1s
    }
```

transition属性

Internet Explorer 10、Firefox、Opera 和 Chrome 支持 transition 属性。 Safari 支持替代的 -webkit-transition 属性。

- transition: property duration timing-function delay;
- transition-property 规定设置过渡效果的 CSS 属性的名称。
- transition-duration 规定完成过渡效果需要多少秒或毫秒。
- transition-timing-function 规定速度效果的速度曲线。
- transition-delay 定义过渡效果何时开始。

（3）2D/3D变换

示例

```
div
{
    transform:rotate(70deg);
    -ms-transform:rotate(70deg);     /* IE 9 */
    -moz-transform:rotate(70deg);    /* Firefox */
    -webkit-transform:rotate(70deg); /* Safari 和 Chrome */
    -o-transform:rotate(70deg);  /* Opera */
}
```

transform属性

- Internet Explorer 10、Firefox、Opera 支持 transform 属性。
- Internet Explorer 9 支持替代的 -ms-transform 属性（仅适用于 2D 转换）。
- Safari 和 Chrome 支持替代的 -webkit-transform 属性（3D 和 2D 转换）。
- Opera 只支持 2D 转换。

重要方法：- translate(x,y) 定义2D转换，向右，向下移动X和Y - translateX(x) - translateY(y) - translateZ(z) - translate3d(x,y,z) 定义3D转换 - scale(x,y) 定义2D缩放转换 - rotate(angle) 定义2D转换 - rotateX(x) - rotateY(y)

- rotateZ(z)

(4)CSS3边框

可以创建圆角边框、添加阴影、使用图片绘制边框 - border-radius - box-shadow - border-image

border-image示例

```
div{
    -webkit-border-image:url(border.png) 30 30 round//依据 图片边框向内偏移 图片边框宽度 图像边框是否拟平铺(repeated/铺满rounded/拉伸stretch
}
```

(5)CSS3背景

background-size属性

规定背景图片的尺寸。CSS3之前是由图片实际尺寸决定的。

```
background:url(i/bg_flower.gif);
background-size:62px 100px;
```

background-origin属性

规定background-position 属性相对于什么位置来定位。

```
background-origin: padding-box|border-box|content-box;
//背景图像相对于内边距框/边框/内容框来定位
```

background-clip属性

规定背景的绘制区域

```
background-clip: border-box|padding-box|content-box;
//背景被裁剪到边框盒/内边距盒/内容框
```

（5）CSS3文本效果

text-shadow 向文本添加阴影

```
text-shadow: h-shadow v-shadow blur color;
```

(6)CSS3字体 ***优先说***

@font-face属性

可以定义自己的字体，不用被迫使用"web-safe"字体 示例

```
<style>
@font-face
{
font-family: myFirstFont;/*font-family定义字体名称*/
src: url('Sansation_Light.ttf'),
     url('Sansation_Light.eot'); /* IE9+ */
font-stretch:  /*定义如何拉伸字体*/
font-weight: '/*定义字体粗细*/
}

div
{
font-family:myFirstFont;//font-family引用字体
}
</style>
```

（7）CSS3选择器

根元素选择器

- ：root{} 总是匹配html

子元素选择器

- ：last-child{}

- p:last-chile{}
- :only-child{}
- :nth-child(n){}

UI伪类选择器

- :enabled 选择启用状态元素
- :disabled 选择禁用状态元素
- :checked 选择被选中的input元素

否定选择器

- :not(<选择器>) 对括号内的选择器取反

3.网页性能优化方法

（1）对全局变量解除引用 ***优先1***

确保占用最少的内存，因为局部变量在离开执行环境时解除引用，对于全局变量和全局变量的属性来说，一旦数据不再用，最好通过将其设置为null来释放其引用。

注意，解除一个值的引用并不意味着自动收回该值所占内存。解除引用的真正作用是让值脱离执行环境，以便垃圾收集器下次运行时将其收回。

（2）优化事件处理程序

事件委托 ***优先2***

因为事件都能冒泡，故尽量在DOM树种高层次节点上添加事件侦听器。通过target(事件真正的目标）的id可以对不同的下层节点做出反应。

因为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，因为首先，每个函数都是对象，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。

移除空事件处理程序

在不需要的时候，移除事件处理程序。内存中有时会残留一些过时不用的"空事件处理程序"，也是造成Web应用程序内存与性能的主要原因。比如有些带有事件处理程序的元素被innerHTML删除了。

（3）作用域

避免全局查找 ***优先3***

在函数内部，如果多次要引用全局对象（比如document），就该起存在函数内部的本地变量(如var doc)中，引用的时候就用本地变量doc.getElementById()来代替document.getElementById()。

避免with语句

with会创建自己的作用域，故会增加其中代码的作用域链的长度。额外的作用域链查找会使with中执行的代码比外面的慢，例如

```
function updateBody(){
    with(document.body){
        alert(tagName);
        innerHTML="Hello World";
    }
}
```

修改为：

```
function updateBody(){
    var body=document.body;
    alert(body.tagName);
    body.innerHTML="Hello World";
}
```

(4)使用正确的方法

避免不必要的属性查找 ***优先4***

变量和数组访问时复杂度O(1)的操作，而对对象的属性查找是复杂度为O(n)的操作。故属性查找越多，执行时间越长。一旦多次用到属性查找，应将其存储在局部变量中，例如

```
var query=window.location.href.substring(window.location.href.indexOf("?"));
```

修改为

```
var url=window.location.href;
var query=url.substring(url.indexOf("?"));
```

优化循环

使用减值迭代， 简化终止条件， 简化循环体， 使用后测试循环（do```while）

展开循环

当循环的次数是确定的，消除循环非对多次函数调用往往更快。

尽量使用原生方法 ***优先5***

原生方法是用C/C++写的，比自己用JS重写的要快，如Math对象中的方法都是原生方法。

Switch语句代替if-else

单个switch语句比一系列复杂运算的if-else快，还可以把case按照最可能到最不可能的顺序进行组织来进一步优化switch语句。

使用位运算符代替其他运算

位运算符比任何布尔运算或算数运算快。选择性地运用位运算替换算数运算可以极大地提升复杂计算的性能。

（5）最小语句话数

多个变量声明改为一个变量声明

```
var count=5;
var color="blue";
var values=
```

[1,2,3];

改为：

```
var count=5,color="blue",values=[1,2,3];
```

使用迭代值时尽量合并语句

```
var name=values[i];
i++;
```

改为

```
var name=values[i++];
```

用对象字面量创建对象代替依次添加属性快 ***优先***

（6）优化DOM交互

最小化现场更新，使用文档片段构建DOM结构

使用innerHTML代替createElement()和appendChild() ***优先***

因为将innerHTML设置为某值时，后台会创建一个HTML解析器，然后使用内部的DOM调用来创建DOM结构，而非基于JavaScript的DOM调用。而内部方法是编译好的简单解释执行的，故要快得多。

事件委托代理

上面已经说了

避免循环体内多次调用HTMLCollection

a.将长度计算放入for循环的初始化部分

```
for(var i=0,len=images.length;i<len;i++){
    ...
}
```

b.把HTMLCollection用变量如images存储,每次循环中添加image=images[]保存当前图像，之后就不用再访问HTMLCollection 了。

返回HTMLCollection对象的情况有：

```
getElementsByTagName()
DOMElement.childNodes
DOMElement.attributes
document.forms、document.images、document.links(包括带href的元素a、area)
```

4.js继承/原型链

原型链

ECMAScript只支持实现继承（不支持接口继承），其实现继承主要是依靠原型链。

原型链简单地讲，就是子类型的原型对象为超类型的实例，即把超类型的实例赋值给了子类型的原型。

类继承/经典继承/伪造对象/借用构造函数

在子类型构造函数的内部调用超类型构造函数。

通过在子类型构造函数内部对超类型构造函数使用 超类型.apply(this)(超类型.call(this))，可以在将来新创建的子类型实例上执行超类型构造函数。故子类型每个实例都会拥有自己的超类型属性副本，即使这个超类型属性是引用类型的，

借用构造函数实例。

```
function SuperType() {
    this.colors=["red","blue","green"];
}

function SubType(args) {
    SuperType.call(this);
}

var instance1=new SubType();
instance1.colors.push("black");
console.log(instance1.colors);//"red,blue,green,black"

var instance2=new SubType();
console.log(instance2.colors);//"red,blue,green"
```

原型继承

借助已有的对象创建新的对象。

不必创建自定义对象，在一个函数内部先创建一个临时性构造函数，然后将传入的对象作为这个构造函数的原型，最后返回这个临时类型的新实例。也就是这个函数对传入其中的对象进行了一次浅复制。包含引用类型值的属性始终都会共享相应的值。

```
function object(o) {
    function F() {

    }
    F.prototype=o;
    return new F();
}

var person={
    name:"Nicholas",
    friends:["Shelby","Court","Van"]
};

var anotherPerson=object(person);
anotherPerson.name="Greg";
anotherPerson.friends.push("Rob");

var yetAnotherPerson=object(person);
yetAnotherPerson.name="Linda";
yetAnotherPerson.friends.push("Barbie");

console.log(person.friends);
```

ECMA5通过新增的Object.create()方法规范了原型继承

9.js闭包

什么是闭包

闭包指有权访问另一个函数作用域的变量的函数，创建闭包的常见方式就是在一个函数内部创建另一个函数，内部函数可以访问包含它的外部函数的参数和变量（除了this和arguments）。

注意闭包的变量

闭包只能取得包含函数中任何变量的最后一个值。

坏处：

闭包会携带包含它的函数的作用域，因此会比其他函数占用更多的内存，过度使用闭包可能会导致内存占用过度。

闭包经典例题：

13）思考下面的代码段：

`for(var i=0;i<5;i++){ var btn=document.createElement('button');//document.createElement(<tag>)方法，创建一个属于指定标签类型的新HTMLElement对象 btn.appendChild(document.createTextNode('Button'+i));//document.createTextNode(<text>)方法，创建一个带有指定内容的新Text对象（即上述button元素上写的文字） btn.addEventListener(//为元素添加事件监听器 'click', function(){ console.log(i); }); document.body.appendChild(btn);//document.A.appendChild(B)方法，将B元素添加为A的子元素 }`（以上方法查看《HTML5权威指南》P589）

a. 点击"Button4"后输出什么？如何使得输出和预期相同

b. 给出一个可以和预期相同的写法。

答案：

a. 输出5，因为形成了闭包，循环结束后，i为5，所有按钮点击都是5

b. 1. 我的方法：

```
```
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    btn.addEventListener(
      'click',
      function(e){
        for(var i=0;i<5;i++){
          if (e.target.innerHTML=='Button'+i) {
            console.log(i);
          }
        }
      }
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法，将B元素添加为A的子元素
}
```
```

作者大神的方法：

这10道javascript笔试题你都会么 中的第 8 题。

参考可以得出其他方法： 2. 闭包

这是错误的：

```
```
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    btn.addEventListener(
      'click',
      (function(i){
        (function(){
          console.log(i);
        })();
      })(i)
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法，将B元素添加为A的子元素
}
```
```

这也是错误的：

```
```
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    btn.addEventListener(
      'click',
      (function(i){
          console.log(i);
      })(i)
    );
    document.body.appendChild(btn);//document.A.appendChild(B)方法，将B元素添加为A的子元素
}
```
```

这才是正确的闭包：

```
```
for(var i=0;i<5;i++){
    var btn=document.createElement('button');
    btn.appendChild(document.createTextNode('Button'+i));
    (function(a){
        btn.addEventListener(
            'click',
            function () {
                console.log(a);
            }
)
    })(i);
    document.body.appendChild(btn);//document.A.appendChild(B)方法，将B元素添加为A的子元素
}
```
```

其实，闭包就是在要引用外部变量i的函数外面加上一个 用作块级作用域的匿名函数

```
```
(function(i){
    //里面内部使用了i的函数
})(i);
```
```

13）引申 这10道javascript笔试题你都会么 中的第 8 题。

实现一段脚本，使得点击对应链接alert出相应的编号

1. DOM 污染法 通过给document元素对象添加了属性值，故污染了DOm

    ``` var lis = document.links;// 属于DOM Document对象，非Dom Element对象,返回文档里具备href属性的a和area元素的对象。参见《HTML5权威指南》P545 for(var i = 0, length = lis.length; i < length; i++) { lis[i].index = i;//此index为自己设置的任意变量值，可任意替换为myindex等等，也可使用拥有的元素对象属性，如id等 lis[i].onclick = function( ) { alert(this.index);//也可用function(e),后面this换为e.target }.

    ```
    }
    ```

    ``` 该法自己的习惯写法为:

    ```
    var lis = document.getElementsByTagName("a");// 属于DOM Document对象，非Dom Element对象 for(var i = 0; i <lis.length; i++) { lis[i].id = i;//把number型的id赋值给id后，直接转换为了string型，因为id都是string型的， lis[i].onclick = function(e) { alert(e.target.id); }; }
    ```

2. 使用闭包

    ``` var lis=document.links;

    for(var i=0,len=lis.length;i

3.我的惯用方法(事件循环变引出*自己命名名*)

```
```
var lis=document.links;

for(var i=0,len=lis.length;i<len;i++){
    lis[i].onclick=function(){
        for(var j=0;j<lis.length;j++){
            if (this==lis[j]) {
                alert(j);
            }
        }
    };
}
```
```

其实，上述i也可就写作i,因为内部循环参数是在局部函数中的，故循环完成后自动销毁，对外部i没有影响。

更多关于闭包其实闭包并不高深莫测

## 10.从地址栏输入域名，说说接下来会发生的事。

详情参见http://www.360doc.com/content/16/0226/14/30980813_537536164.shtml

**1）在浏览器里输入网址**

例如我的数据新闻网站weiryer.com

**2）浏览器根据DNS系统查找域名对应的主机IP地址**
```

具体来讲，查找过程如下：

（1）浏览器缓存

浏览器会缓存DNS记录一段时间。

（2）系统缓存

在浏览器缓存没有找到需要的记录，浏览器就会做一个系统调用（windows里是gethostbyname),获得系统缓存中的记录。

（5）路由器缓存

接着查询请求发向路由器，路由器一般也有自己的DNS缓存

（6）ISP的DNS缓存

接下来，是查找互联网移动提供商的DNS服务器。一般都能找到相应的缓存记录。

(7)递归搜索

ISP的DNS服务器从.com顶级域名服务器到具体的域名服务器。

DNS还里令人新视的是，facebook.com这样的域名看上去只对应一个单独的IP地址，所以效率问题是个瓶颈。解决办法是，- **DNS循环里用**DNS负载均衡是通过循环复用实现的，**地理DNS**能根据用户所在的地理位置，通过把域名映射到多个不同的IP地址以提高可扩展性。- **Anycast**是一个IP地址映射到多个物理主机的路由技术。大多数DNS服务器使用Anycast来获得高效低延迟的DNS查找。

3）浏览器给web服务器发送一个HTTP请求。

因为很多页面都是动态页面（比如facebook、QQ空间），就算浏览器缓存中有，打开浏览器缓存也是过期了，所有不能从中读取，所以浏览器要把请求发送到facebook所在的服务器。

请求例如下面这个：

```
GET http://facebook.com/ HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml, [...]
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; [...]
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: facebook.com
Cookie: datr=1265876274-[...]; locale=en_US; lsd=WW[...]; c_user=2101[...]
```

GET 这个请求定义了要读取的URL，“http://facebook.com”。浏览器自身定义 (User-Agent 头)，和它希望接受什么类型的响应 (Accept and Accept-Encoding 头)。Connection头要求服务器为了后边的请求不要关闭TCP连接，请求中也包含浏览器存储的该域名的cookies。

你fiddler、FireBug这样的工具都可以用来查看原始HTTP请求。Chrome也可以查看当前网页的HTTP头http://blog.sina.com.cn/s/blog_809ff62a01018jzq.html

4)facebook网站服务器的永久重定向

facebook服务器发回给浏览器的响应如下：

```
HTTP/1.1 301 Moved Permanently
Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Location: http://www.facebook.com/
P3P: CP="DSP LAW"
Pragma: no-cache
Set-Cookie: made_write_conn=deleted; expires=Thu, 12-Feb-2009 05:09:50 GMT;
path=/; domain=.facebook.com; httponly
Content-Type: text/html; charset=utf-8
X-Cnection: close
Date: Fri, 12 Feb 2010 05:09:51 GMT
Content-Length: 0
```

响应的是一个301的永久重定向，就是把网址补全了，访问http://www.facebook.com/而非http://facebook.com/

为什么服务器一定要重定向呢？

答案有。

1.搜索引擎排名有关，如果一个页面有两个地址，就像http://www.igoro.com/ 和http://igoro.com/，搜索引擎会认为它们是两个网站，结果造成每一个的搜索链接指数都会减少。

2.不同的网址会造成缓存友好性变差，一个页面有好几个名字的话，它可能在缓存里出现好几次。

5)浏览器跟踪重定向地址

现在，浏览器知道了“http://www.facebook.com/”才是要访问的正确地址，所以它会发送另一个获取请求。

```
GET http://www.facebook.com/ HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml, [...]
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; [...]
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Cookie: lsd=XW[...]; c_user=21[...]; x-referer=[...]
Host: www.facebook.com
```

6)服务器处理请求

服务器接收到请求，开始处理。

a.比如映射网址地址结构的文件层次存储，，像http://example.com/folder1/page1.aspx这个地 址会映射为htpdocs/folder1/page1.aspx这个文件。web服务器软件可以设置成为地址人工的对应请求处理，这样 page1.aspx的发布地址就可以是http://example.com/folder1/page1。

b.请求处理阅读请求及它的参数和cookies。它会读取也可能更新一些数据，并将数据存储在服务器上。然后，需求处理会生成一个HTML响应。

7）服务器发回一个HTML响应。

```
HTTP/1.1 200 OK
Cache-Control: private, no-store, no-cache, must-revalidate, post-check=0,
pre-check=0
Expires: Sat, 01 Jan 2000 00:00:00 GMT
P3P: CP="DSP LAW"
Pragma: no-cache
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
X-Cnection: close
Transfer-Encoding: chunked
Date: Fri, 12 Feb 2010 09:05:55 GMT

2b37m@[...]
```

报头中把Content-type设置为“text/html”。报头让浏览器将该响应内容以HTML形式呈现，而不是以文件形式下载它。

8）浏览器开始显示HTML

在浏览器没有完整接受全部HTML文档时，它就已经开始显示这个页面了。

9)浏览器请求获取HTML中的对象

比如img中的图片的地址等，以类似的过程查找域名、发送请求、重定向等等。

包不像动态页面那样，静态文件会允许浏览器对其进行缓存。有的文件可能会不需要与服务器通讯，而从缓存中直接读取。服务器的响应中包含了静态文件保存的期限 信息，所以浏览器知道需要它们缓存多长时间。

静态内容也能通过CDN（内容分发网络）轻松的复制。通常网站会使用第三方的CDN。例如，Facebook的静态文件由最大的CDN提供商Akamai来托管。

10）浏览器发送异步AJAX请求。

知识点补充：

DNS（Domain Name System，域名系统），因特网上作为域名和IP地址相互映射的一个分布式数据库，能够使用户更方便的访问互联网，而不用去记住能够被机器直接读取的IP数串。通过主机名，最终得到该主机名对应的IP地址的过程叫做域名解析（或主机名解析）。

ISP(Internet Service Provider)，互联网移动提供商，即向广大用户综合提供互联网接入业务、信息业务、和增值业务的电信运营商。例如三大基础运营商，电信，移动，联通，

小区局域网宽带接入方式，是国内大中城市目前最普及的一种宽带接入方式，互联网服务提供商（ISP）采用光纤接入到楼（FTTB），再通过建立小区局域网同网线接入用户室，为整幢楼或小区提供共享带宽（通常是10-100兆/秒）。

11.Ajax原理和过程

Ajax原理简介

Ajax是异步JavaScript和XML的缩写。该名称诞生于XML还是数据传输首选格式的时期，现在这种情况已经不复存在。

Ajax技术的核心是XMLHttpRequest对象（简称XHR）。XHR为向服务器发送请求和解析服务器响应提供了底畅的接口。能够以异步方式从服务器取得更多信息，意味着用户单击后，不用刷新页面也能获得新数据。即可以使用XHR对象取得新数据。然后通过DOM将新数据插入到页面中。

Ajax好处：(1)可以异步生成请求。用户可以在表单被处理时继续与文档交互。而传统提交表单发送到服务器，用户必须等待服务器处理整数据并生成响应。(2)响应信息可以仅更新页面的一部分。而传统提交表单到服务器后，整个页面被刷新，所有上下文信息都丢失了。

Ajax过程

GET实例

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Ajax</title>
    </head>
    <body>
        <div>
            <button id="btn">Apples</button>

        </div>
        <div id="target">

        </div>
        <script>
                var button=document.getElementById("btn");
                button.onclick=handleButtonPress;

                function handleButtonPress(e) {
                    var xhr=new XMLHttpRequest();//创建XMLHttpRequest对象

                    xhr.onreadystatechange=function(){
                        if (xhr.readyState==4) {//xhr.readyState:该属性表示请求/响应过程的当前活动阶段，4为完成，表明现在一次都会触发一次
                            if (xhr.status>=200&&xhr.status<300)||xhr.status==304) {//xhr.status：响应的状态
                                document.getElementById("target").innerHTML=xhr.responseText;//xhr.responseText：作为响应主体被返回的
                            }
                            else{
                                document.getElementById("target").innerHTML="Request was unsuccessful: "+xhr.status;
                            }
                        }
                    }

                    xhr.open("GET","example.html",true);//要发送的类型、请求的URL和是否异步发送

                    xhr.send(null);//发送作为请求主体要发送的数据

                }
        </script>
    </body>
</html>
```

点击按钮后，具体写法可总结为：

```
- (1)var xhr=new XMLHttpRequest() 创建XMLHttpRequest对象
- (2)指定xhr.onreadystatechange事件函数，在函数中先检测readyState属性（表示请求/响应的阶段）为4（表完成），再检测相应状态(xhr.status>=20
- (3)xhr.open("GET", URL,是否异步)
- (4)xhr.send(null)
```

POST实例(by json方法)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ajax2</title>
    </head>
```

```
    <body>
        <form id="fruitform" method="post" action="http://www.wetryer.com:3000/form"><!--点击按钮后跳转向页面http://127.0.0.1:8080/fo
            <p>Bnanas:</p>
            <p><input name="bananas" value="2"/></p>
            <p>Apples:</p>
            <p><input name="apples" value="5"/></p>
            <p>Cherries:</p>
            <p><input name="cherries" value="20"/></p>

            <p>Total:</p>
            <div id="results">0 items</div>
            <button id="submit" type="submit">Submit Forme/button>
        </form>
    </body>
    <script>
        document.getElementById("submit").onclick=handleButtonPress;
        var xhr;
        function handleButtonPress(e) {
            e.preventDefault();//取消按钮提交表单的默认行为
            var form=document.getElementById("fruitform");


            var formData=new Object();
            var inputElements=document.getElementsByTagName("input");
            for (var i=0;i<inputElements.length;i++) {
                formData[inputElements[i].name]=inputElements[i].value;
            }//将表单提交的数据转换为obJect类型


            xhr=new XMLHttpRequest();//创建XMLHttpRequest()对象
            xhr.onreadystatechange=handleResponse;//为xhr.onreadystatechange设置事件监听函数，readystatechange改变的时候触发
            xhr.open("post",form.action,true);//xhr.open()设置post方法，发送到的URL，是异步
            xhr.setRequestHeader("Content-Type","application/json");//设置请求的首部字段Content-Type为application/json,告诉服务器你发送
            xhr.send(JSON.stringify(formData));//浏览器向服务器发送数据前，要把数据从Object转换为JSON字符串
        }

        function handleResponse() {
            if (xhr.readyState==4) {
                if (xhr.status>=200&&xhr.status<300||xhr.status==304) {
                    var data=JSON.parse(xhr.responseText);//把JSON数据解析为对象
                    document.getElementById("results").innerHTML="You ordered "+data.total+" items";
                }
            }

        }
    </script>
</html>
```

点击按钮后，具体写法可总结为：

```
    (1) e.preventDefault()取消按钮提交表单的默认行为
    (2) 获取表单数据，并能将转换为Object类型
    (3)xhr=new XMLHttpRequest() 创建XMLHttpRequest对象
    (4)指定xhr.onreadystatechange事件函数，在函数中先检测readyState4属性为4，再检测相应状态(xhr.status>=200&&xhr.status<300)||xhr.status==
    (5)xhr.open("POST", URL,是否异步)
    (6)xhr.setRequestHeader("Content-Type","application/json")设置请求的首部字段Content-Type为application/json,告诉服务器你发送json数据
    (7) xhr.send(JSON.stringify(formData))浏览器向服务器发送数据前，要把数据从Object转换为JSON字符串
```

POST实例（by 表单默认编码方式）

html部分网上例，此处仅列出javascript部分。

```
    <script>
        document.getElementById("submit").onclick=handleButtonPress;
        var xhr;
        function handleButtonPress(e) {
            e.preventDefault();//取消按钮提交表单的默认行为
            var form=document.getElementById("fruitform");

            var formData="";
            var inputElements=document.getElementsByTagName("input");
            for (var i=0;i<inputElements.length;i++) {
                formData=inputElements[i].name+"="+inputElements[i].value+"&";
            }//将表单提交的数据转换为编码表单数据的默认方式（application/x-www-form-urlencoded编码）。


            xhr=new XMLHttpRequest();//创建XMLHttpRequest()对象
            xhr.onreadystatechange=handleResponse;//为xhr.onreadystatechange设置事件监听函数，readystatechange改变的时候触发
            xhr.open("post",form.action,true);//xhr.open()设置post方法，发送到的URL，是异步
            xhr.setRequestHeader("Content-Type","application/x-www-form-urlencoded");//设置请求的首部字段Content-Type为application/ja
            xhr.send(formData);//浏览器向服务器发送数据前，要把数据从Object转换为JSON字符串
        }

        function handleResponse() {
            if (xhr.readyState==4) {
                if (xhr.status>=200&&xhr.status<300||xhr.status==304) {
                    document.getElementById("results").innerHTML=xhr.responseText;
                }
            }

        }
    </script>
```

12.http协议了解

http://www.cnblogs.com/li0803/archive/2008/11/03/1324746.html ***复习《图解HTTP》***

（1）HTTP协议相关网络基础

HTTP（HyperText Transfer Protocol，超文本传输协议）是基于请求与响应模式的、无状态的、位于七层网络协议中应用层的协议。常基于TCP的连接方式，HTTP1.1版本中给出了一种持续连接的方式。

HTTP于1990年问世，1997年公布的HTTP1.1是目前最流行的版本。

TCP/IP协议族

网络是在TCP/IP协议族的，HTTP是其内部的一个子集。

网络传输七层协议是，应该会传网数物。

应用层：
TCP/IP协议族的HTTP、FTP(文件传输协议）、DNS(域名系统）处于应用层。

传输层：
TCP/IP协议族的TCP（传输控制协议）、UDP(用户数据报协议）处于该层。

网络层:
规定了数据怎样的路径到达对方计算机，并把数据包传给对方。 TCP/IP协议族的IP（网际协议）在该层。

数据链路层：
处理网络连接的硬件部分，包括操作系统、硬件设备驱动、网卡等。

负责传输IP协议，与HTTP密切相关的协议之一

IP协议的作用是把数据包传送给对方。需要IP地址（节点被分配到的地址）和Mac地址（网卡所属的固定地址）。通信中转时利用中转站Mac地址。传输路线通过路由机制。

确保可靠的TCP协议，与HTTP密切相关的协议之二

TCP提供对靠的字节流服务，把大数据分割成报文段为单位的数据包，并能够确认数据最终是否送达了对方。

TCP三次握手策略

- 第一次，发送端发送带SYN(synchronize，同步)标志的数据包给对方。
- 第二次，接收端收到后，传回一个带有SYNACK标志的数据包。
- 第三次，发送端再回传一个带ACK标志的数据包，代表"握手"结束。

DNS服务

位于应用层，负责域名解析，提供通过域名查找IP地址，或逆向从IP地址反查域名的服务。

HTTP通信过程各协议作用

```
a.（客户端）客户端输入页面地址
b.DNS，按域名解析为ip地址
c.HTTP协议，生成针对目标Web服务器的HTTP请求报文。
d.TCP协议，为了通信方便，将HTTP请求报文分为多个报文段，并通过二次握手将每个报文段可靠地传给对方。
e.IP协议，查找对方的地址，一层层传一边传送出去。
f.（服务器端）TCP协议，通过二次握手从对方那里接收到报文段，并按序号以用来的顺序重组请求报文。
g.HTTP协议，对收请求的服务器上的内容进行处理。
h.请求的处理结果也同样利用TCP/IP协议传给客户端
```

URL和URI

URI:统一资源标识符。
URL:统一资源定位符。
URI除了包含URL还包含与URN(统一资源名称，命名资源，但不知如何定位）。所以，每个URL都是URI，URI不一定是URL。

（2）HTTP协议概述

HTTP是无状态协议

HTTP协议自身不对请求和响应之间的通信状态进行保存，即不保存之前的一切请求或响应报文的信息。

目的，为了更快地大量处理事务，所以要这么简单。

为了解决保持状态的技术，cookie技术。

HTTP方法

GET:获取资源

请求被URI识别的资源，并返回响应内容。

POST:传输数据

在已标识的资源后附加新数据，并接受数据的处理结果。

PUT:传输文件

传输文件到URI指定的位置

HEAD:获得报文首部

同GET，但不返回报文主体，只返回响应报文首部。

DELETE:删除文件

与PUT相反的方法，因其不带验证机制故一般不能使用，当遵守REST标准或配合Web验证机制时可能会开放。

OPTIONS:询问支持的方法

查询针对请求的URI指定资源支持的方法，返回结果可能是GET，POST,HEAD,OPTIONS。

TRACE:追踪路径

让服务器返回收到的请求信息给客户端。可查询发送出的请求是怎样被加工或篡改的，主要用于测试或诊断。

CONNECT:要求用隧道协议连接代理

要求在与代理服务器通信时建立隧道，实现用隧道协议进行TCP通信。

持久连接节省通信量

非模式

HTTP从期服本是每进行一次HTTP通信都要先建立TCP连接，然后再断开。模式是，建立TCP连接→HTTP请求/响应→断开TCP连接→建立TCP连接→HTTP请求/响应→断开TCP连接……

新模式

HTTP1.1和部分HTTP1.0有了**持久连接**的方法，只要任意一端没有明确提出断开连接，则保持TCP连接状态。模式是，建立TCP连接→HTTP请求/响应……→HTTP请求/响应→`'''`→断开TCP连接

管线化

持久连接使多数请求以管线化方式发送成为可能。以前是发送请求后等待并接受到响应才能发送下一个请求，现在是不用等待响应也能发送下一个请求。

（3）HTTP报文

请求报文

包括：a.报文首部 (空行) b.报文主体
a.报文首部包括：
a.1请求行。
包含用于请求的方法，请求行构成，方法+ URI+ 协议版本。
a.2首部字段（请求首部字段，通用首部字段，实体首部字段）
a.3其他，比如cookie。

响应报文

包括：a.报文首部 b.空行 c.报文主体
a.报文首部包括：
a.1状态行。
包含表明请求结果的状态码，状态行构成，协议版本+ 状态码+ 状态码的原因短语。
a.2首部字段（响应首部字段，通用首部字段，实体首部字段）
a.3其他，比如cookie。

编码提升传输效率

实体主体

通常，报文主体等于实体主体。当传输中进行编码操作时，实体主体内容发生了变化。

内容编码

应用在实体主体上的编码格式，并保持实体主体信息原样压缩。内容编码后的实体由客户端接收并解码。

分块传输编码

HTTP通信中，请求的编码实体资源未完全传输之前，浏览器无法显示请求页面。传输大容量数据时，通过把数据分割成多块，可以让浏览器逐步显示页面。

可发送多多种数据的多部分对象集合

HTTP协议中采纳了多部分对象集合，发送的报文中可以含多类型实体。 需在首部字段追加多个Content-type和Content-Range

可获取部分内容

为避免下载过程中遇到网络中断重头开始的情况，需要一种可恢复机制，即从之前中断处恢复下载。

要实现该功能需要指定下载的实体范围，用首部字段Range来指定资源的byte范围,例如bytes=50001~5000

内容协商返回最适合的内容

内容协商机制是客户端和服务器端就响应的资源内容进行交涉，然后提供给客户端最为合适的资源。

包含在请求报文中的一些首部字段（Accept,Accept-Charset,Accept-Encoding,Accept-Language,Content-language)就是内容协商判断的标准。

（4）HTTP状态码

表示客户端HTTP请求的返回结果、标记服务器端是否处理正常、通知出现的错误等工作。

状态码类别：

- 1XX （信息性状态码）接收的请求正在处理
- 2XX （成功状态码）请求正常处理完毕
- 3XX（重定向状态码）需要进行附加操作以完成请求
- 4XX （客户端错误状态码）请求有语法错误等导致服务器无法处理请求
- 5XX （服务器错误状态码）服务器处理请求出错

2XX

- 200 OK //请求处理成功
- 204 No Content//服务器处理成功，但无内容返回
- 206 Partial Content//客户端进行了范围请求，服务器成功处理了这部分GET请求。

3XX

- 301 Moved Permanently//永久重定向。表请求的资源已被分配了新的URL加指定资源路径最后没有加/
- 302 Found//临时重定向。
- 303 See Other//同302，但明确表示客户端应当采用GET方法获取资源。
- 304 Not Modified//未修改，所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。即重定向无先，可以从缓存加载。

4XX

- 400 Bad Request//客户端请求报文中有语法错误
- 401 Unauthorized//发送的请求需要通过HTTP认证
- 403 Forbidden//服务器理解此请求，但拒绝执行
- 404 Not Found//服务器无法根据客户端的请求找到资源

5XX

- 500 Internal Server Error//服务器执行请求时内部发生了错误
- 503 Service Unavailable//服务器由于超载或系统维护，现在无法完成请求。

(7)通信数据转发程序

代理

客户端和服务器的中间人。接受客户端发送的请求转发给服务器，转发时附加Via首部字段以标记经经过的主机信息。

网关

能使通信线路上的服务器提供非HTTP协议服务，利用网关能提高通信的安全性。

隧道

通过隧道的传输，可以和远距离的服务器进行通信。

（8）确保Web安全的HTTPS

HTTP+SSL(安全套接层）加密+认证+完整性保护=HTTPS

12.缓存实现（ css,js等文件）

HTML5appchache简介

HTML5的应用缓存（application cache,简称appcache），专门为开发离线Web应用而设计，appcache就是从浏览器的缓存中分出来的一块缓存区。

不支持HTML5应用缓存的浏览器有IE,比较旧的FireFox。比较旧的Safari，其他都支持。

appcache使用方法

（1）使用描述文件列出要下载和缓存的资源

示例描述文件：

```
CACHE MANIFEST
#Comment

file.js
file.css
```

文件扩展名以前推荐.manifest,现在推荐.appcache。

（2）将描述文件与页面关联起来

```
<html manifest="/offline.manifest">
```

为html标签设置manifest属性指定该描述文件路径。

（3）应用缓存的更新和启用

applicationCache对象

JavaScript有一个applicationCache对象，其有一个表示status的属性，表示应用缓存的当前状态， applicationCache.status的值及其对应状态如下：

```
0，无缓存，即没有与页面相关的缓存
1，闲置，即应用缓存未得到更新
2，检查中，即正在下载描述文件并检查更新
3，下载中，即应用缓存中附下载描述文件中指定的资源
4，更新完成，即应用缓存已经更新的资源，且新资源都下载完毕，可以通过swapCatch()来使用了。
5，废弃，即应用缓存的描述文件已经不在了，最页面无法再访问应用缓存。
```

应用缓存还有很多相关的事件。

自动/手动检查更新应用缓存

自动，一般来讲，随着页面加载,applicationCache对象的事件会依次触发，即应用缓存会自动检查更新下载缓存资源。

手动，使用applicationCache.update()方法可以手工干预，可以让应用缓存为了检查更新而触发上述事件。

```
applicationCache.update()
```

调用update()后，应用缓存就会去检查描述文件是否更新（触发checking事件），然后就像页面刚刚加载一样，继续执行后续操作。
后面如果触发了cached事件，说明应用缓存已经准备就绪，不再发生其他操作了。
如果触发了updateready事件，说明新版本缓存已经可用，此时应调用swapCache()来启用新应用缓存。

```
Event.util.addHandler(applicationCache,"updateready",function(){
    applicationCache.swapCache();
});
```

13.如何实现新闻网站标题的实时更新

（1）Ajax

可以用个定时器，每隔一段时间用Ajax向网站发送GET请求，获取新的标题

（2）WebSocket

WebSocket是一种新浏览器API，可跨域，可在一个单独的持久连接上提供全双工、双向通信。

由于WebSocket使用了自定义的协议而非HTTP协议，故能够在客户端和服务器之间发送非常少量的数据，不必担心HTTP那样字节级的开销，但它必须要求是专门的Web Socket服务器。

实例：

```
var socket=new WebSocket("ws://www.example.com/server.php");
socket.send("Hello world!");
socket.onmessage=function(event){
    var data=event.data;
    //处理数据，可以用这些数据更新页面的某部分
}
```

（3）SSE

SSE API用于创建到服务器的单向连接，服务器通过这个连接可以发送任意数量的数据，SSE是通过常规的HTTP通信。如果只需读取服务器，则可以选择SSE，SSE和Ajax结合起来也可以实现双通信。

14.JQuery链式操作

就是可以在一句代码里面对同一个DOM对象实现不同的操作。非链式的话就要每做一次操作就获取一次对象，链式只用获取一次对象。

实例

```
$("#mydiv").addClass("current");//给当前元素添加"current"样式
.next().show();//获取下一个元素并显示
.parent().siblings().children("a").removeClass("current");//获取父元素的兄弟元素的子元素<a>并移除它的"current"样式
.next().hide();//隐藏下一个元素
```

15.浏览器兼容性

1.事件冒泡方式

所有现代浏览器都支持冒泡，IE5.5之前事件冒泡跳过html(从body直接u跳到document)。其他IE9、Firefox、Chrome和Safari则将事件一直冒泡到windows.

2.事件捕获支持度

老版本的浏览器不支持捕获，故最好都用冒泡。

3.三种事件处理程序

- IE事件处理程序attachEvent()和detachEvent()只有两个参数，因为IE8及之前版本只支持冒泡，且attachEvent()和detachEvent()的第一个参数带on("onclick"),DOM2级不带on("click")。
- IE事件处理程序在全局作用域运行，不像DOM2级的和DOM0级的方法是在所属元素的作用域运行，其中的this===window.
- 在同一个事件上添加多个addEventListener(),或多个attachEvent()，addEventListener()顺序执行，IE事件处理程序attachEvent()是反过来的顺序执行的。
- DOM0级对每个事件只支持一个事件处理程序，DOM2和IE事件处理程序都支持多个。
- 支持IE事件处理程序的浏览器有IE和Opera。
- 支持DOM2级的有IE9、FF、Safari、Chrome、Opera。
- 支持DOM0的兼容浏览器。（所以IE浏览器只有IE9及对DOM2）

4.IE的事件对象与DOM中的event对象及其属性的不同*P358*

- IE在使用DOM0级方法添加事件处理程序时，要通过window.event获得event。IE使用其他方法则与DOM中的event相同。
- IE的event.srcElement和DOM的event.target属性相同
- IE取消默认事件方法:

```
event.returnValue =false;
```

其他取消默认事件方法:

```
event.preventDefault();
```

- IE的cancelBubble属性和DOM的stopPropagation()作用类似。 event.cancelBuble=true只能停止事件冒泡；而event.stopPropagation()则能停止冒泡和捕获。

认真反复看EventUtil对象的写法！！

```
var EventUtil={
    addHandler:function(element,type,handler){
        if (element.addEventListener) {
            element.addEventListener(type,handler,false);//表示在冒泡阶段调用事件处理程序
        }
        else if (element.attachEvent) {//IE8及之前版本
            element.attachEvent(type,handler);
        }
        else{
            element["on"+event]=handler;
        }
    },
    removeHandler:function(element,type,handler){
        if (element.removeEventListener) {
            element.removeEventListener(type,handler,false);
        }
        else if (element.detachEvent) {
            element.detachEvent("on"+type,handler);
        }
        else{
            element["on"+type]=null;
        }
    },
    getEvent:function(event){
        return event? event:window.event;
    },
    getTarget:function(event){
        return event.target||event.srcElement;
    },
    preventDefault:function(event){
        if (event.preventDefault) {
            event.preventDefault();
        }
        else if(event.returnValue){
            event.returnValue =false;
        }
    },
    getRelatedTarget:function(event){
        if (event.relatedTarget) {
            return event.relatedTarget;
        }
        else if (event.toElement) {//IE8之前只有下面两种，IE9三种都有
            return event.toElement;
        }
        else if (event.fromElement) {
            return event.fromElement;
        }
        else{
            return null;
        }
    },
    getButton: function(event){
        if (document.implementation.hasFeature("MouseEvents","2.0")) {
            return event.button;//DOM将event.button赋三个值，0，1，2
        }
        else{
            switch (event.button) {//IE8之前的很复杂，加下列表
                case 0:
                case 1:
                case 3:
                case 5:
                case 7:
                    return 0;
                case 2:
                case 6:
                    return 2;
                case 4:
                    return 1;
            }
        }
    },
    getWheelDelta:function(event){
        if (event.wheelDelta) {//大多数浏览器的mousewhee1事件，及相应event.wheelDelta属性
            return(client.engine.opera&&client.engine.opera<9.5? -event.wheelDelta:event.wheelDelta);
        }
        else{
            return -event.detail*40;//Firefox及DOMMouseScroll事件，相应滚动信息在event.detail属性
        }
    },
    getCharCode: function(event){
        if (typeof event.charCode=="number") {
            return event.charCode;
        }
        else{
            return event.keyCode;
        }
    },
    getClipboardText:function(event){
        var clipboardData=(event.clipboardData||window.clipboardData);
        return clipboardData.getData("text");
    },
    setClipboardText: function(event,value){
        if (event.clipboardData) {
            return event.clipboardData.setData("text/plain",value);
        }
        else if (window.clipboardData) {
            return window.clipboardData.setData("text",value);
        }
    }
};
```

16.

两个非常大的整数相加，整数大到计算机的整型数据已经无法保存了，要求写一个函数来进行计算。

```
  var a="12345678";
  var b="22334455";
  var len=Math.max(a.length,b.length);
  var relAdvance=0;
  var result=[];


  for (var i=0;i<len;i++) {
      if (i==0) {
          aData=Number(a.slice(-1));
          bData=Number(b.slice(-1));
      }
      else{
          if(a.slice(-1-i,-i)){
              aData=Number(a.slice(-1-i,-i));
          }
          else{
              aData=0
          }

          if(b.slice(-1-i,-i)){
              bData=Number(b.slice(-1-i,-i));
          }
          else{
              bData=0;
          }



      }



      console.log(aData);
      console.log(bData);
      var rel=aData+bData+Number(relAdvance);
      console.log(rel);
      var relCurrent=String(rel).slice(-1);
      console.log(relCurrent);
      if (relCurrent.length>1) {
          var relAdvance=String(rel).slice(0,1);
      }
      else{
          var relAdvance=0;
      }
      console.log(relAdvance);
      result.unshift(relCurrent);
  }
  console.log(result);
  var resultStr=result[0];
  for (var i=1,l=result.length;i<l;i++) {
      var resultStr=resultStr.concat(result[i]);
  }

  console.log(resultStr);
```

17.

自定义右键菜单

方法一，使用oncontextmenu事件，所有浏览器都支持 oncontextmenu 事件， contextmenu 元素只有 Firefox 浏览器支持。

```
<!DOCTYPE html>

<html>
<head>
    <title>Page Title</title>
    <style type="text/css">
        #YaWindow{
            position: absolute;
            width: 150px;
            height: 150px;
            background-color: lightgray;
            display: none;
            border:thin solid blue;
        }
        ul{
            position: absolute;
            margin: 0px;
            padding: 0px;
            width: 100%;
```

```
            height: 100%;
            border: thin solid red;
            list-style-type: none;/*去掉li前面点*/
        }
        li{
            position: absolute;
            margin: 0px;
            padding: 0px;
            left: 0px;
            height: 50px;
            width: 100%;
            text-align: center;
            vertical-align: middle;
            border: thin solid black;
        }
        ul :nth-child(1){
            top: 0px;
        }
        ul :nth-child(2){
            top:50px;
        }
        ul :nth-child(3){
            top: 100px;
        }



    </style>
</head>

<body>
    <div id="myDiv">text here</div>

    <div id="rWindow">
        <ul>
            <li><p>love</p></li>
            <li><p>like</p></li>
            <li><p>value</p></li>
        </ul>

    </div>
    <script src="prepare.js"></script>
</body>
</html>
```

javascript:

```
```//EventUtil对象
var div=document.getElementById("myDiv");

EventUtil.addHandler(div,"contextmenu",function(event){
    event=EventUtil.getEvent(event);
    EventUtil.preventDefault(event);

    var rWindowLeft=event.clientX-"px";
    var rWindowTop=event.clientY+"px";
    var rWindow=document.getElementById("rWindow");

    rWindow.style.setProperty("left",rWindowLeft);
    rWindow.style.setProperty("top",rWindowTop);
    rWindow.style.setProperty("display","block");

});
EventUtil.addHandler(document,"click",function(event){
    event=EventUtil.getEvent(event);
     var rWindow=document.getElementById("rWindow");
        rWindow.style.setProperty("display","none");

})
```

方法二：检测mouseenter的event.button===2

不能屏蔽掉oncontextmenu事件带来的默认菜单，故该方法并不可取

```
var div=document.getElementById("myDiv");

EventUtil.addHandler(div,"mousedown",function(event){
    event=EventUtil.getEvent(event);
    var buttonNum=EventUtil.getButton(event);
    if (buttonNum==2) {
        EventUtil.preventDefault(event);//这样好像屏蔽不掉原右键菜单，只能求助oncontextmenu事件
        var rWindowLeft=event.clientX+"px";
        var rWindowTop=event.clientY+"px";
        var rWindow=document.getElementById("rWindow");

        rWindow.style.setProperty("left",rWindowLeft);
        rWindow.style.setProperty("top",rWindowTop);
        rWindow.style.setProperty("display","block");

    }
});
EventUtil.addHandler(document,"click",function(event){
    event=EventUtil.getEvent(event);
     var rWindow=document.getElementById("rWindow");
        rWindow.style.setProperty("display","none");

})
```

18.XSS

跨站脚本攻击(Cross Site Scripting)，为不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆，故将跨站脚本攻击缩写为XSS。恶意攻击者往Web页面里插入恶意Script代码，当用户浏览该页之时，嵌入其中Web里面的Script代码会被执行，从而达到恶意攻击用户的特殊目的。

XSS漏洞是Web应用程序中最常见的漏洞之一，如果您的站点没有预防XSS漏洞的固定方法，那么就存在XSS漏洞，这个利用XSS漏洞的病毒之所以具有重要意义是因为，通常难以查到XSS漏洞的威胁，而该病毒则是其发挥得淋漓尽致。

19.CSRF

CSRF（Cross-site request forgery跨站请求伪造，也被称为"One Click Attack"或者Session Riding，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。尽管听起来像跨站脚本（XSS），但它与XSS非常不同，并且攻击方式几乎相左。XSS利用站点内的信任用户，而CSRF则通过伪装来自受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。

20.nodejs

总述

nodejs是后端JavaScript的运行平台,它保留了前端浏览器JavaScript中那些熟悉的接口，没有改写语言本身的任何特性，依旧基于作用域和原型链，区别在与它将前端的思想迁移到了后端。

nodejs的特点

（1）异步I/O

在node中，可以自然地进行并行I/O操作。

非阻塞I/O

nodejs利用并步，让单线程**远离阻塞**，以更好地使用CPU。又利用单线程，远离多线程死锁、状态同步等问题。

轮询

阻塞I/O造成CPU等待浪费，但非阻塞I/O带来的麻烦事需要**轮询**去确认是否完全完成数据获取，会让CPU处理状态判断，是对CPU资源的浪费。

（2）事件和回调函数编程

Nodejs将前端浏览器应用广泛的事件引入后端，配合异步I/O，将事件点暴露给业务逻辑。事件的编程方式具有轻量级、松耦合、只关注事务点等优势。

回调函数

回调函数也是一大特色，是最好的接受异步调用返回数据的方式。

事件循环——node自身的执行模型

事件循环使得回调函数十分普遍，进程启动时，Node便会创建一个类似于while(true)的循环，每执行一次循环体的过程称为Tick。每个Tick过程就是查看是否有事件等待处理，如果有，就取出事件及其相关回调函数。如果存在关联的回调函数，就执行它们。

（3）单线程

Node保持了JavaScript单线程的特点，在Node中，JavaScript与其余线程是无法共享任何状态的。单线程不用像多线程那样处处在意状态的同步问题，没有死锁的存在，也没有线程上下文交换所带来的任何性能上的开销。

（4）跨平台

Nodejs兼容Linux和Windows。

（5）模块机制 *important*

nodejs应用场景

适合I/O密集型和CPU密集型。

看自己写的网站的Nodejs

21.跨域

1.CORS技术

CORS(Cross-Origin Resource Sharing,跨源资源共享）是定义了在必须访问跨源资源时，浏览器与服务器应该如何沟通。其基本思想，就是使用自定义的HTTP头，让浏览器和服务器进行沟通，从而决定请求或响应是否应该成功。

做法是，

在请求首部加上应该Origin字段,包含请求页面的源信息（协议、域名和端口）。

```
Origin:http://www.nczonline.net
```

如果服务器认为这个请求可以接受，就在在响应首部加上Access-Control-Allow-Origin首部字段，在其中写上相同的信息。

```
Access-Control-Allow-Origin:http//www.nczonline.net
```

如果没有该首部字段，或该首部字段跟信息不匹配，浏览器就会驳回请求。

2.各浏览器对CORS的实现

IE对CORS的实现：XDR

XDR与AJAX的XHR类似，但都实现安全可靠的跨域通信，用法也类似。并能缓解CSRF和XSS。

其他浏览器对CORS的实现:XHR

在open()方法中传入绝对URL。但它是有限制的。

2.图像Ping

标签

该标签完全不用担心是否跨域

创建动态图像的图像Ping

图像Ping与服务器进行简单的、单向的跨域通信的一种方式。

```
var img=new Image();
img.onload=img.onerror=function(){
    alert("Done!");
}
img.src="http://www.example.com/test?name=Nicholas";
```

图像 Ping只能发送GET请求，且无法访问服务器响应文本。
```

3.JSONP(见22.)

4.Comet

Ajax是浏览器页面向服务器请求数据的技术。Comet是服务器向页面推送数据的技术。

两种实现Comet的方式：长轮询和流。

5.SSE

SSE(Server-Sent Events,服务器发送事件)，SSE API用于创建到服务器的单向连接，服务器通过这个连接可以发送任意数量的数据。SSE是通过常规的HTTP通信，如果只需读取服务器，则可以选择SSE。SSE和Ajax结合起来也可以实现双通信。

6.WebSocket

WebSocket是一种新浏览器API，可跨域，可在一个单独的持久连接上提供全双工、双向通信。

由于WebSocket使用了自定义的协议而非HTTP协议，故能够在客户端和服务器之间发送非常少量的数据，不必担心HTTP那样字节级的开销。但它必须要求是专门的Web Socket服务器。

实例。

```
var socket=new WebSocket("ws://www.example.com/server.php");
socket.send("Hello world!");
socket.onmessage=function(event){
    var data=event.data;
    //处理数据，可以用这些数据更新页面的某部分
}
```

22.JSONP原理

参见http://kb.cnblogs.com/page/139725/ - 1、一个众所周知的问题，Ajax直接请求普通文件存在跨域无权限访问的问题，甭管你是静态页面、动态网页、web服务、WCF，只要是跨域请求，一律不准。- 2、不过我们又发现，Web页面上调用js文件时则不受是否跨域的影（不仅如此，我们还发现凡是拥有"src"这个属性的标签都拥有跨域的能力，比如