$javascript编程部分

请用原生js实现一个函数,给页面制定的任意一个元素添加一个透明遮罩(透明度可变,默认0.2),使这个区域点击无效,要求兼容IE8+及各主流浏览器,遮罩层效果如下图所示:

```
<style>
#target {
    width: 200px;
    height: 300px;
    margin: 40px;
    background-color: tomato;
}
</style>

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

<script>
function addMask(elem, opacity) {
    opacity = opacity || 0.2;

    var rect = elem.getBoundingClientRect();
    var style = getComputedStyle(elem, null);

    var mask = document.createElement('div');
    mask.style.position = 'absolute';
    var marginLeft = parseFloat(style.marginLeft);
    mask.style.left = (elem.offsetLeft - marginLeft) +
'px';
    var marginTop = parseFloat(style.marginTop);
    mask.style.top = (elem.offsetTop - marginTop) + 'p
x';
    mask.style.zIndex = 9999;
    mask.style.opacity = '' + opacity;
    mask.style.backgroundColor = '#000';

    mask.style.width = (parseFloat(style.marginLeft) +
        parseFloat(style.marginRight) + rect.width) +
'px';
    mask.style.height = (parseFloat(style.marginTop) +
        parseFloat(style.marginBottom) + rect.height) +
 'px';

    elem.parentNode.appendChild(mask);
}

var target = document.getElementById('target');
```

```
addMask(target);

target.addEventListener('click', function () {
    console.log('click');
}, false);
</script>
```

请用代码写出(今天是星期x)其中x表示当天是星期几,如果当天是星期一,输出应该是”今天是星期一”

```
var days = ['日','一','二','三','四','五','六'];
var date = new Date();

console.log('今天是星期' + days[date.getDay()]);
```

下面这段代码想要循环延时输出结果0 1 2 3 4,请问输出结果是否正确,如果不正确,请说明为什么,并修改循环内的代码使其输出正确结果

```
for (var i = 0; i < 5; ++i) {
  setTimeout(function () {
    console.log(i + ' ');
  }, 100);
}
```

不能输出正确结果，因为循环中setTimeout接受的参数函数通过闭包访问变量i。javascript运行环境为单线程，setTimeout注册的函数需要等待线程空闲才能执行，此时for循环已经结束，i值为5.五个定时输出都是5
修改方法：将setTimeout放在函数立即调用表达式中，将i值作为参数传递给包裹函数，创建新闭包

```
for (var i = 0; i < 5; ++i) {
  (function (i) {
    setTimeout(function () {
      console.log(i + ' ');
    }, 100);
  }(i));
}
```

现有一个Page类,其原型对象上有许多以post开头的方法(如postMsg);另有一拦截函数chekc,只返回ture或false.请设计一个函数,该函数应批量改造原Page的postXXX方法,在保留其原有功能的同时,为每个postXXX方法增加拦截验证功能,当chekc返回true时继续执行原postXXX方法,返回false时不再执行原postXXX方法

```
function Page() {}

Page.prototype = {
  constructor: Page,

  postA: function (a) {
    console.log('a:' + a);
  },
  postB: function (b) {
    console.log('b:' + b);
  },
  postC: function (c) {
    console.log('c:' + c);
  },
  check: function () {
    return Math.random() > 0.5;
  }
}

function checkfy(obj) {
  for (var key in obj) {
    if (key.indexOf('post') === 0 && typeof obj[key] ==
= 'function') {
      (function (key) {
        var fn = obj[key];
        obj[key] = function () {
          if (obj.check()) {
            fn.apply(obj, arguments);
          }
        };
      }(key));
    }
  }
} // end checkfy()

checkfy(Page.prototype);

var obj = new Page();

obj.postA('checkfy');
obj.postB('checkfy');
```

```
41.  obj.postC('checkfy');
```

编写javascript深度克隆函数deepClone

```
function deepClone(obj) {
    var _toString = Object.prototype.toString;

    // null, undefined, non-object, function
    if (!obj || typeof obj !== 'object') {
        return obj;
    }

    // DOM Node
    if (obj.nodeType && 'cloneNode' in obj) {
        return obj.cloneNode(true);
    }

    // Date
    if (_toString.call(obj) === '[object Date]') {
        return new Date(obj.getTime());
    }

    // RegExp
    if (_toString.call(obj) === '[object RegExp]') {
        var flags = [];
        if (obj.global) { flags.push('g'); }
        if (obj.multiline) { flags.push('m'); }
        if (obj.ignoreCase) { flags.push('i'); }

        return new RegExp(obj.source, flags.join(''));
    }

    var result = Array.isArray(obj) ? [] :
        obj.constructor ? new obj.constructor() : {};

    for (var key in obj ) {
        result[key] = deepClone(obj[key]);
    }

    return result;
}

function A() {
    this.a = a;
}
```

```
42.
43. var a = {
44.     name: 'qiu',
45.     birth: new Date(),
46.     pattern: /qiu/gim,
47.     container: document.body,
48.     hobbys: ['book', new Date(), /aaa/gim, 111]
49. };
50.
51. var c = new A();
52. var b = deepClone(c);
53. console.log(c.a === b.a);
54. console.log(c, b);
```

补充代码,鼠标单击Button1后将Button1移动到Button2的后面

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>TEst</title>
</head>
<body>

<div>
   <input type="button" id ="button1" value="1" />
   <input type="button" id ="button2" value="2" />
</div>

<script type="text/javascript">
    var btn1 = document.getElementById('button1');
    var btn2 = document.getElementById('button2');

    addListener(btn1, 'click', function (event) {
        btn1.parentNode.insertBefore(btn2, btn1);
    });

    function addListener(elem, type, handler) {
        if (elem.addEventListener) {
            elem.addEventListener(type, handler,
false);
            return handler;
        } else if (elem.attachEvent) {
            function wrapper() {
                var event = window.event;
                event.target = event.srcElement;
                handler.call(elem, event);
            }
            elem.attachEvent('on' + type, wrapper);
            return wrapper;
        }
    }

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

网页中实现一个计算当年还剩多少时间的倒数计时程序,要求网页上实时动态显示”××年还剩××天××时××分××秒”

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>TEst</title>
</head>
<body>

    <span id="target"></span>


<script type="text/javascript">
    // 为了简化。每月默认30天
    function getTimeString() {
        var start = new Date();
        var end = new Date(start.getFullYear() + 1, 0, 1);
        var elapse = Math.floor((end - start) / 1000);

        var seconds = elapse % 60 ;
        var minutes = Math.floor(elapse / 60) % 60;
        var hours = Math.floor(elapse / (60 * 60)) % 24;
        var days = Math.floor(elapse / (60 * 60 * 24)) % 30;
        var months = Math.floor(elapse / (60 * 60 * 24 * 30)) % 12;
        var years = Math.floor(elapse / (60 * 60 * 24 * 30 * 12));

        return start.getFullYear() + '年还剩' + years + '年' + months + '月' + days + '日'
            + hours + '小时' + minutes + '分' + seconds + '秒';
    }

    function domText(elem, text) {
        if (text == undefined) {

            if (elem.textContent) {
                return elem.textContent;
```

```
            } else if (elem.innerText) {
                return elem.innerText;
            }
        } else {
            if (elem.textContent) {
                elem.textContent = text;
            } else if (elem.innerText) {
                elem.innerText = text;
            } else {
                elem.innerHTML = text;
            }
        }
    }

    var target = document.getElementById('target');

    setInterval(function () {
        domText(target, getTimeString());
    }, 1000)
</script>

</body>
</html>
```

完成一个函数,接受数组作为参数,数组元素为整数或者数组,数组元素包含整数或数组,函数返回扁平化后的数组

如：[1, [2, [[3, 4], 5], 6]] => [1, 2, 3, 4, 5, 6]

```
    var data =  [1, [2, [ [3, 4], 5], 6]];

    function flat(data, result) {
        var i, d, len;
        for (i = 0, len = data.length; i < len; ++i) {
            d = data[i];
            if (typeof d === 'number') {
                result.push(d);
            } else {
                flat(d, result);
            }
        }
    }

    var result = [];
    flat(data, result);

    console.log(result);
```

如何判断一个对象是否为数组

如果浏览器支持Array.isArray()可以直接判断否则需进行必要判断

```
/**
 * 判断一个对象是否是数组，参数不是对象或者不是数组，返回false
 *
 * @param {Object} arg 需要测试是否为数组的对象
 * @return {Boolean} 传入参数是数组返回true，否则返回false
 */
function isArray(arg) {
    if (typeof arg === 'object') {
        return Object.prototype.toString.call(arg) === '[object Array]';
    }
    return false;
}
```

请评价以下事件监听器代码并给出改进意见

```
1.  if (window.addEventListener) {
2.    var addListener = function (el, type, listener, useCa
    pture) {
3.      el.addEventListener(type, listener, useCapture);
4.    };
5.  }
6.  else if (document.all) {
7.    addListener = function (el, type, listener) {
8.      el.attachEvent('on' + type, function () {
9.        listener.apply(el);
10.     });
11.   };
12. }
```

作用：浏览器功能检测实现跨浏览器DOM事件绑定

优点：

1. 测试代码只运行一次，根据浏览器确定绑定方法
2. 通过`listener.apply(el)`解决IE下监听器this与标准不一致的地方
3. 在浏览器不支持的情况下提供简单的功能，在标准浏览器中提供捕获功能

缺点：

1. document.all作为IE检测不可靠，应该使用if(el.attachEvent)
2. addListener在不同浏览器下API不一样
3. `listener.apply`使this与标准一致但监听器无法移除
4. 未解决IE下listener参数event。 target问题

改进：

```
var addListener;

if (window.addEventListener) {
  addListener = function (el, type, listener, useCapture) {
    el.addEventListener(type, listener, useCapture);
    return listener;
  };
}
else if (window.attachEvent) {
  addListener = function (el, type, listener) {
    // 标准化this, event, target
    var wrapper = function () {
      var event = window.event;
      event.target = event.srcElement;
      listener.call(el, event);
    };

    el.attachEvent('on' + type, wrapper);
    return wrapper;
    // 返回wrapper。调用者可以保存，以后remove
  };
}
```

如何判断一个对象是否为函数

```
/**
 * 判断对象是否为函数，如果当前运行环境对可调用对象（如正则表达式）
 * 的typeof返回'function'，采用通用方法，否则采用优化方法
 *
 * @param {Any} arg 需要检测是否为函数的对象
 * @return {boolean} 如果参数是函数，返回true，否则false
 */
function isFunction(arg) {
    if (arg) {
        if (typeof (/./) !== 'function') {
            return typeof arg === 'function';
        } else {
            return Object.prototype.toString.call(arg) === '[object Function]';
        }
    } // end if
    return false;
}
```

编写一个函数接受url中query string为参数,返回解析后的Object,query string使用application/x-www-form-urlencoded编码

```
/**
 * 解析query string转换为对象，一个key有多个值时生成数组
 *
 * @param {String} query 需要解析的query字符串，开头可以是?,
 * 按照application/x-www-form-urlencoded编码
 * @return {Object} 参数解析后的对象
 */
function parseQuery(query) {
    var result = {};

    // 如果不是字符串返回空对象
    if (typeof query !== 'string') {
        return result;
    }

    // 去掉字符串开头可能带的?
    if (query.charAt(0) === '?') {
        query = query.substring(1);
    }

    var pairs = query.split('&');
    var pair;
    var key, value;
    var i, len;

    for (i = 0, len = pairs.length; i < len; ++i) {
        pair = pairs[i].split('=');
        // application/x-www-form-urlencoded编码会将' '转换为+
        key = decodeURIComponent(pair[0]).replace(/\+/g, ' ');
        value = decodeURIComponent(pair[1]).replace(/\+/g, ' ');

        // 如果是新key，直接添加
        if (!(key in result)) {
            result[key] = value;
        }
        // 如果key已经出现一次以上，直接向数组添加value
        else if (isArray(result[key])) {
```

```
            result[key].push(value);
        }
        // key第二次出现，将结果改为数组
        else {
            var arr = [result[key]];
            arr.push(value);
            result[key] = arr;
        } // end if-else
    } // end for

    return result;
}

function isArray(arg) {
    if (arg && typeof arg === 'object') {
        return Object.prototype.toString.call(arg) === '[object Array]';
    }
    return false;
}
/**
console.log(parseQuery('sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8'));
 */
```

解析一个完整的url,返回Object包含域与window.location相同

```
/**
 * 解析一个url并生成window.location对象中包含的域
 * location:
 * {
 *      href: '包含完整的url',
 *      origin: '包含协议到pathname之前的内容',
 *      protocol: 'url使用的协议，包含末尾的:',
 *      username: '用户名', // 暂时不支持
 *      password: '密码',  // 暂时不支持
 *      host: '完整主机名，包含:和端口',
 *      hostname: '主机名，不包含端口'
 *      port: '端口号',
 *      pathname: '服务器上访问资源的路径/开头',
 *      search: 'query string，?开头',
 *      hash: '#开头的fragment identifier'
 * }
 *
 * @param {string} url 需要解析的url
 * @return {Object} 包含url信息的对象
 */
function parseUrl(url) {
    var result = {};
    var keys = ['href', 'origin', 'protocol', 'host',
                'hostname', 'port', 'pathname', 'searc
h', 'hash'];
    var i, len;
    var regexp = /(([^:]+:)\/\/(([^:\/\?#]+)(:\d+)?))
(\/[^?#]*)?(\?[^#]*)?(#.*)?/;

    var match = regexp.exec(url);

    if (match) {
        for (i = keys.length - 1; i >= 0; --i) {
            result[keys[i]] = match[i] ? match[i] : '';
        }
    }

    return result;
}
```

完成函数getViewportSize返回指定窗口的视口尺寸

```
/**
* 查询指定窗口的视口尺寸，如果不指定窗口，查询当前窗口尺寸
**/
function getViewportSize(w) {
    w = w || window;

    // IE9及标准浏览器中可使用此标准方法
    if ('innerHeight' in w) {
        return {
            width: w.innerWidth,
            height: w.innerHeight
        };
    }

    var d = w.document;
    // IE 8及以下浏览器在标准模式下
    if (document.compatMode === 'CSS1Compat') {
        return {
            width: d.documentElement.clientWidth,
            height: d.documentElement.clientHeight
        };
    }

    // IE8及以下浏览器在怪癖模式下
    return {
        width: d.body.clientWidth,
        height: d.body.clientHeight
    };
}
```

完成函数getScrollOffset返回窗口滚动条偏移量

```
/**
 * 获取指定window中滚动条的偏移量，如未指定则获取当前window
 * 滚动条偏移量
 *
 * @param {window} w 需要获取滚动条偏移量的窗口
 * @return {Object} obj.x为水平滚动条偏移量,obj.y为竖直滚动条偏移量
 */
function getScrollOffset(w) {
    w =  w || window;
    // 如果是标准浏览器
    if (w.pageXOffset != null) {
        return {
            x: w.pageXOffset,
            y: w.pageYOffset
        };
    }

    // 老版本IE，根据兼容性不同访问不同元素
    var d = w.document;
    if (d.compatMode === 'CSS1Compat') {
        return {
            x: d.documentElement.scrollLeft,
            y: d.documentElement.scrollTop
        }
    }

    return {
        x: d.body.scrollLeft,
        y: d.body.scrollTop
    };
}
```

现有一个字符串richText,是一段富文本,需要显示在页面上.有个要求,需要给其中只包含一个img元素的

p标签增加一个叫pic的class.请编写代码实现.可以使用jQuery或KISSY.

```
function richText(text) {
    var div = document.createElement('div');
    div.innerHTML = text;
    var p = div.getElementsByTagName('p');
    var i, len;

    for (i = 0, len = p.length; i < len; ++i) {
        if (p[i].getElementsByTagName('img').length === 1) {
            p[i].classList.add('pic');
        }
    }

    return div.innerHTML;
}
```

请实现一个Event类,继承自此类的对象都会拥有两个方法on,off,once和trigger

```
function Event() {
    if (!(this instanceof Event)) {
        return new Event();
    }
    this._callbacks = {};
}
Event.prototype.on = function (type, handler) {
    this_callbacks = this._callbacks || {};
    this._callbacks[type] = this.callbacks[type] || [];
    this._callbacks[type].push(handler);

    return this;
};

Event.prototype.off = function (type, handler) {
    var list = this._callbacks[type];

    if (list) {
        for (var i = list.length; i >= 0; --i) {
            if (list[i] === handler) {
                list.splice(i, 1);
            }
        }
    }

    return this;
};

Event.prototype.trigger = function (type, data) {
    var list = this._callbacks[type];

    if (list) {
        for (var i = 0, len = list.length; i < len; ++
i) {
            list[i].call(this, data);
        }
    }
};

Event.prototype.once = function (type, handler) {
    var self = this;
```

```
41.
42.     function wrapper() {
43.         handler.apply(self, arguments);
44.         self.off(type, wrapper);
45.     }
46.     this.on(type, wrapper);
47.     return this;
48. };
```

编写一个函数将列表子元素顺序反转

```
1.  <ul id="target">
2.      <li>1</li>
3.      <li>2</li>
4.      <li>3</li>
5.      <li>4</li>
6.  </ul>
7.
8.  <script>
9.      var target = document.getElementById('target');
10.     var i;
11.     var frag = document.createDocumentFragment();
12.
13.     for (i = target.children.length - 1; i >= 0; --i) {
14.         frag.appendChild(target.children[i]);
15.     }
16.     target.appendChild(frag);
17. </script>
```

以下函数的作用是?空白区域应该填写什么

```
// define
(function (window) {
    function fn(str) {
        this.str = str;
    }

    fn.prototype.format = function () {
        var arg = __1__;
        return this.str.replace(__2__, function (a, b) {
            return arg[b] || '';
        });
    };

    window.fn = fn;
})(window);

// use
(function () {
    var t = new fn('<p><a href="{0}">{1}</a><span>{2}</span></p>');
    console.log(t.format('http://www.alibaba.com', 'Alibaba', 'Welcome'));
})();
```

define部分定义一个简单的模板类，使用{}作为转义标记，中间的数字表示替换目标，format实参用来替换模板内标记
横线处填:

1. `Array.prototype.slice.call(arguments, 0)`
2. `/\{\s*(\d+)\s*\}/g`

编写一个函数实现form的序列化(即将一个表单中的键值序列化为可提交的字符串)

```
<form id="target">
    <select name="age">
        <option value="aaa">aaa</option>
        <option value="bbb" selected>bbb</option>
    </select>
    <select name="friends" multiple>
        <option value="qiu" selected>qiu</option>
        <option value="de">de</option>
        <option value="qing" selected>qing</option>
    </select>
    <input name="name" value="qiudeqing">
    <input type="password" name="password" value="11111">
    <input type="hidden" name="salery" value="3333">
    <textarea name="description">description</textarea>
    <input type="checkbox" name="hobby" checked value="football">Football
    <input type="checkbox" name="hobby" value="basketball">Basketball
    <input type="radio" name="sex" checked value="Female">Female
    <input type="radio" name="sex" value="Male">Male
</form>


<script>

/**
 * 将一个表单元素序列化为可提交的字符串
 *
 * @param {FormElement} form 需要序列化的表单元素
 * @return {string} 表单序列化后的字符串
 */
function serializeForm(form) {
  if (!form || form.nodeName.toUpperCase() !== 'FORM') {
    return;
  }

  var result = [];

```

```
37.    var i, len;
38.    var field, fieldName, fieldType;
39.
40.    for (i = 0, len = form.length; i < len; ++i) {
41.      field = form.elements[i];
42.      fieldName = field.name;
43.      fieldType = field.type;
44.
45.      if (field.disabled || !fieldName) {
46.        continue;
47.      } // enf if
48.
49.      switch (fieldType) {
50.        case 'text':
51.        case 'password':
52.        case 'hidden':
53.        case 'textarea':
54.          result.push(encodeURIComponent(fieldName) + '='
     +
55.              encodeURIComponent(field.value));
56.          break;
57.
58.        case 'radio':
59.        case 'checkbox':
60.          if (field.checked) {
61.            result.push(encodeURIComponent(fieldName) +
    '=' +
62.                encodeURIComponent(field.value));
63.          }
64.          break;
65.
66.        case 'select-one':
67.        case 'select-multiple':
68.          for (var j = 0, jLen = field.options.length; j
     < jLen; ++j) {
69.            if (field.options[j].selected) {
70.              result.push(encodeURIComponent(fieldName) +
     '=' +
71.                  encodeURIComponent(field.options[j].value
     || field.options[j].text));
72.            }
```

```
            } // end for
            break;

        case 'file':
        case 'submit':
            break; // 是否处理?

        default:
            break;
      } // end switch
    } // end for

    return result.join('&');
}

var form = document.getElementById('target');
console.log(serializeForm(form));
</script>
```

使用原生javascript给下面列表中的li节点绑定点击事件,点击时创建一个Object对象,兼容IE和标准浏览器

```
<ul id="nav">
    <li><a href="http://11111">111</a></li>
    <li><a href="http://2222">222</a></li>
    <li><a href="http://333">333</a></li>
    <li><a href="http://444">444</a></li>
</ul>

Object:
{
    "index": 1,
    "name": "111",
    "link": "http://1111"
}
```

script:

```
var EventUtil = {
    getEvent: function (event) {
        return event || window.event;
    },
    getTarget: function (event) {
        return event.target || event.srcElement;
    },
    // 返回注册成功的监听器，IE中需要使用返回值来移除监听器
    on: function (elem, type, handler) {
        if (elem.addEventListener) {
            elem.addEventListener(type, handler, false);
            return handler;
        } else if (elem.attachEvent) {
            function wrapper(event) {
                return handler.call(elem, event);
            };
            elem.attachEvent('on' + type, wrapper);
            return wrapper;
        }
    },
    off: function (elem, type, handler) {
        if (elem.removeEventListener) {
            elem.removeEventListener(type, handler, false);
        } else if (elem.detachEvent) {
            elem.detachEvent('on' + type, handler);
        }
    },
    preventDefault: function (event) {
        if (event.preventDefault) {
            event.preventDefault();
        } else if ('returnValue' in event) {
            event.returnValue = false;
        }
    },
    stopPropagation: function (event) {
        if (event.stopPropagation) {
            event.stopPropagation();
        } else if ('cancelBubble' in event) {
            event.cancelBubble = true;
```

```
        }
    }
};
var DOMUtil = {
    text: function (elem) {
        if ('textContent' in elem) {
            return elem.textContent;
        } else if ('innerText' in elem) {
            return elem.innerText;
        }
    },
    prop: function (elem, propName) {
        return elem.getAttribute(propName);
    }
};

var nav = document.getElementById('nav');

EventUtil.on(nav, 'click', function (event) {
    var event = EventUtil.getEvent(event);
    var target = EventUtil.getTarget(event);

    var children = this.children;
    var i, len;
    var anchor;
    var obj = {};

    for (i = 0, len = children.length; i < len; ++i) {
        if (children[i] === target) {
            obj.index = i + 1;
            anchor = target.getElementsByTagName('a')
[0];
            obj.name = DOMUtil.text(anchor);
            obj.link = DOMUtil.prop(anchor, 'href');
        }
    }

    alert('index: ' + obj.index + ' name: ' + obj.name
 +
        ' link: ' + obj.link);
});
```

有一个大数组,var a = [‘1’, ‘2’, ‘3’, …];a的长度是100,内容填充随机整数的字符串.请先构造此数组a,然后设计一个算法将其内容去重

```
    /**
    * 数组去重
    **/
    function normalize(arr) {
        if (arr && Array.isArray(arr)) {
            var i, len, map = {};
            for (i = arr.length; i >= 0; --i) {
                if (arr[i] in map) {
                    arr.splice(i, 1);
                } else {
                    map[arr[i]] = true;
                }
            }
        }
        return arr;
    }

    /**
    * 用100个随机整数对应的字符串填充数组。
    **/
    function fillArray(arr, start, end) {
        start = start == undefined ? 1 : start;
        end = end == undefined ?  100 : end;

        if (end <= start) {
            end = start + 100;
        }

        var width = end - start;
        var i;
        for (i = 100; i >= 1; --i) {
            arr.push('' + (Math.floor(Math.random() * w
idth) + start));
        }
        return arr;
    }

    var input = [];
    fillArray(input, 1, 100);
    input.sort(function (a, b) {
        return a - b;
```

```
41.     });
42.     console.log(input);
43.
44.     normalize(input);
45.     console.log(input);
```