$javascript概念部分

DOM元素e的e.getAttribute(propName)和e.propName有什么区别和联系

- e.getAttribute()，是标准DOM操作文档元素属性的方法，具有通用性可在任意文档上使用，返回元素在源文件中**设置的属性**
- e.propName通常是在HTML文档中访问特定元素的**特性**，浏览器解析元素后生成对应对象（如a标签生成HTMLAnchorElement），这些对象的特性会根据特定规则结合属性设置得到，对于没有对应特性的属性，只能使用getAttribute进行访问
- e.getAttribute()返回值是源文件中设置的值，类型是字符串或者null（有的实现返回”“）
- e.propName返回值可能是字符串、布尔值、对象、undefined等
- 大部分attribute与property是一一对应关系，修改其中一个会影响另一个，如id，title等属性
- 一些布尔属性`<input hidden/>`的检测设置需要hasAttribute和removeAttribute来完成，或者设置对应property
- 像`<a href="../index.html">link</a>`中href属性，转换成property的时候需要通过转换得到完整URL
- 一些attribute和property不是一一对应如：form控件中`<input value="hello"/>`对应的是defaultValue，修改或设置value property修改的是控件当前值，setAttribute修改value属性不会改变value property

offsetWidth/offsetHeight,clientWidth/clientHeight与scrollWidth/scrollHeight的区别

- offsetWidth/offsetHeight返回值包含**content + padding + border**，效果与e.getBoundingClientRect()相同
- clientWidth/clientHeight返回值只包含**content + padding**，如果有滚动条，也**不包含滚动条**
- scrollWidth/scrollHeight返回值包含**content + padding + 溢出内容的尺寸**

XMLHttpRequest通用属性和方法

1. `readyState`:表示请求状态的整数，取值:
    - UNSENT（0）：对象已创建
    - OPENED（1）：open()成功调用，在这个状态下，可以为xhr设置请求头，或者使用send()发送请求
    - HEADERS_RECEIVED(2)：所有重定向已经自动完成访问，并且最终响应的HTTP头已经收到
    - LOADING(3)：响应体正在接收
    - DONE(4)：数据传输完成或者传输产生错误

2. `onreadystatechange`：readyState改变时调用的函数
3. `status`：服务器返回的HTTP状态码（如，200， 404）
4. `statusText`:服务器返回的HTTP状态信息（如，OK，No Content）
5. `responseText`:作为字符串形式的来自服务器的完整响应
6. `responseXML`: Document对象，表示服务器的响应解析成的XML文档
7. `abort()`:取消异步HTTP请求
8. `getAllResponseHeaders()`: 返回一个字符串，包含响应中服务器发送的全部HTTP报头。每个报头都是一个用冒号分隔开的名/值对，并且使用一个回车/换行来分隔报头行
9. `getResponseHeader(headerName)`:返回headName对应的报头值
10. `open(method, url, asynchronous [, user, password])`:初始化准备发送到服务器上的请求。method是HTTP方法，不区分大小写；url是请求发送的相对或绝对URL；asynchronous表示请求是否异步；user和password提供身份验证
11. `setRequestHeader(name, value)`:设置HTTP报头
12. `send(body)`:对服务器请求进行初始化。参数body包含请求的主体部分，对于POST请求为键值对字符串；对于GET请求，为null

focus/blur与focusin/focusout的区别与联系

1. focus/blur不冒泡，focusin/focusout冒泡
2. focus/blur兼容性好，focusin/focusout在除FireFox外的浏览器下都保持良好兼容性，如需使用事件托管，可考虑在FireFox下使用事件捕获elem.addEventListener('focus', handler, true)
3. 可获得焦点的元素:
    1. window

2. 链接被点击或键盘操作
3. 表单空间被点击或键盘操作
4. 设置`tabindex`属性的元素被点击或键盘操作

mouseover/mouseout与mouseenter/mouseleave的区别与联系

1. mouseover/mouseout是标准事件，**所有浏览器都支持**；mouseenter/mouseleave是IE5.5引入的特有事件后来被DOM3标准采纳，现代标准浏览器也支持
2. mouseover/mouseout是**冒泡**事件；mouseenter/mouseleave**不冒泡**。需要为**多个元素监听鼠标移入/出事件时，推荐mouseover/mouseout托管，提高性能**
3. 标准事件模型中event.target表示发生移入/出的元素,**vent.relatedTarget**对应移出/如元素；在老IE中event.srcElement表示发生移入/出的元素，**event.toElement**表示移出的目标元素，**event.fromElement**表示移入时的来源元素

例子：鼠标从div#target元素移出时进行处理，判断逻辑如下：

```
<div id="target"><span>test</span></div>

<script type="text/javascript">
var target = document.getElementById('target');
if (target.addEventListener) {
  target.addEventListener('mouseout', mouseoutHandler, false);
} else if (target.attachEvent) {
  target.attachEvent('onmouseout', mouseoutHandler);
}

function mouseoutHandler(e) {
  e = e || window.event;
  var target = e.target || e.srcElement;

  // 判断移出鼠标的元素是否为目标元素
  if (target.id !== 'target') {
    return;
  }

  // 判断鼠标是移出元素还是移到子元素
  var relatedTarget = event.relatedTarget || e.toElement;
  while (relatedTarget !== target
    && relatedTarget.nodeName.toUpperCase() !== 'BODY') {
    relatedTarget = relatedTarget.parentNode;
  }

  // 如果相等，说明鼠标在元素内部移动
  if (relatedTarget === target) {
    return;
  }

  // 执行需要操作
  //alert('鼠标移出');

}
</script>
```

sessionStorage,localStorage,cookie区别

1. 都会在浏览器端保存，有大小限制，同源限制
2. cookie会在请求时发送到服务器，作为会话标识，服务器可修改cookie；web storage不会发送到服务器
3. cookie有path概念，子路径可以访问父路径cookie，父路径不能访问子路径cookie
4. 有效期：cookie在设置的有效期内有效，默认为浏览器关闭；sessionStorage在窗口关闭前有效，localStorage长期有效，直到用户删除
5. 共享：sessionStorage不能共享，localStorage在同源文档之间共享，cookie在同源且符合path规则的文档之间共享
6. localStorage的修改会促发其他文档窗口的update事件
7. cookie有secure属性要求HTTPS传输
8. 浏览器不能保存超过300个cookie，单个服务器不能超过20个，每个cookie不能超过4k。web storage大小支持能达到5M

javascript跨域通信

同源：两个文档同源需满足

1. 协议相同
2. 域名相同
3. 端口相同

跨域通信：js进行DOM操作、通信时如果目标与当前窗口不满足同源条件，浏览器为了安全会阻止跨域操作。跨域通信通常有以下方法

- 如果是log之类的简单**单项通信**，新建`<img>`,`<script>`,`<link>`,`<iframe>`元素，通过src，href属性设置为目标url。实现跨域请求
- 如果请求**json数据**，使用`<script>`进行jsonp请求
- 现代浏览器中**多窗口通信**使用HTML5规范的targetWindow.postMessage(data, origin);其中data是需要发送的对象，origin是目标窗口的origin。window.addEventListener('message', handler, false);handler的event.data是postMessage发送来的数据，event.origin是发送窗口的origin，event.source是发送消息的窗口引用
- 内部服务器代理请求跨域url，然后返回数据

- 跨域请求数据，现代浏览器可使用HTML5规范的CORS功能，只要目标服务器返回HTTP头部`Access-Control-Allow-Origin: *`即可像普通ajax一样访问跨域资源

javascript有哪几种数据类型

六种基本数据类型

- undefined
- null
- string
- boolean
- number
- symbol(ES6)

一种引用类型

- Object

什么闭包,闭包有什么用

闭包是在某个作用域内定义的函数，它可以访问这个作用域内的所有变量。闭包作用域链通常包括三个部分:

1. 函数本身作用域。
2. 闭包定义时的作用域。
3. 全局作用域。

闭包常见用途:

1. 创建特权方法用于访问控制
2. 事件处理程序及回调

javascript有哪几种方法定义函数

1. 函数声明表达式
2. function操作符
3. Function 构造函数

4. ES6:arrow function

重要参考资料：MDN:Functions_and_function_scope

应用程序存储和离线web应用

HTML5新增应用程序缓存，允许web应用将应用程序自身保存到用户浏览器中，用户离线状态也能访问。
1.为html元素设置manifest属性:`<html manifest="myapp.appcache">`，其中后缀名只是一个约定，真正识别方式是通过`text/cache-manifest`作为MIME类型。所以需要配置服务器保证设置正确
2.manifest文件首行为`CACHE MANIFEST`，其余就是要缓存的URL列表，每个一行，相对路径都相对于manifest文件的url。注释以#开头
3.url分为三种类型：`CACHE`:为默认类型。`NETWORK`：表示资源从不缓存。`FALLBACK`:每行包含两个url，第二个URL是指需要加载和存储在缓存中的资源， 第一个URL是一个前缀。任何匹配该前缀的URL都不会缓存，如果从网络中载入这样的URL失败的话，就会用第二个URL指定的缓存资源来替代。以下是一个文件例子：

```
CACHE MANIFEST

CACHE:
myapp.html
myapp.css
myapp.js

FALLBACK:
videos/ offline_help.html

NETWORK:
cgi/
```

客户端存储localStorage和sessionStorage

- localStorage有效期为永久，sessionStorage有效期为顶层窗口关闭前
- 同源文档可以读取并修改localStorage数据，sessionStorage只允许同一个窗口下的文档访问，如通过iframe引入的同源文档。

- Storage对象通常被当做普通javascript对象使用：**通过设置属性来存取字符串值**，也可以通过**setItem(key, value)设置**，**getItem(key)读取**，**removeItem(key)删除**，**clear()删除所有数据**，**length表示已存储的数据项数目**，**key(index)返回对应索引的key**

```
localStorage.setItem('x', 1); // storge x->1
localStorage.getItem('x'); // return value of x

// 枚举所有存储的键值对
for (var i = 0, len = localStorage.length; i < len; ++i
 ) {
    var name = localStorage.key(i);
    var value = localStorage.getItem(name);
}

localStorage.removeItem('x'); // remove x
localStorage.clear();  // remove all data
```

cookie及其操作

- cookie是web浏览器存储的少量数据，最早设计为服务器端使用，作为HTTP协议的扩展实现。cookie数据会自动在浏览器和服务器之间传输。
- 通过读写cookie检测是否支持
- cookie属性有**名**，**值**，**max-age**，**path**, **domain**，**secure**；
- cookie默认有效期为浏览器会话，一旦用户关闭浏览器，数据就丢失，通过设置**max-age=seconds**属性告诉浏览器cookie有效期
- cookie作用域通过**文档源**和**文档路径**来确定，通过**path**和**domain**进行配置，web页面同目录或子目录文档都可访问
- 通过cookie保存数据的方法为：为document.cookie设置一个符合目标的字符串如下
- 读取document.cookie获得’; ‘分隔的字符串，key=value,解析得到结果

```
document.cookie = 'name=qiu; max-age=9999; path=/; domain=domain; secure';

document.cookie = 'name=aaa; path=/; domain=domain; secure';
// 要改变cookie的值，需要使用相同的名字、路径和域，新的值
// 来设置cookie，同样的方法可以用来改变有效期

// 设置max-age为0可以删除指定cookie

//读取cookie，访问document.cookie返回键值对组成的字符串，
//不同键值对之间用'; '分隔。通过解析获得需要的值
```

cookieUtil.js：自己写的cookie操作工具

javascript有哪些方法定义对象

1. 对象字面量：`var obj = {};`
2. 构造函数：`var obj = new Object();`
3. Object.create(): `var obj = Object.create(Object.prototype);`

===运算符判断相等的流程是怎样的

1. 如果两个值不是相同类型，它们不相等
2. 如果两个值都是null或者都是undefined，它们相等
3. 如果两个值都是布尔类型true或者都是false，它们相等
4. 如果其中有一个是**NaN**，它们不相等
5. 如果都是数值型并且数值相等，他们相等， -0等于0
6. 如果他们都是字符串并且在相同位置包含相同的16位值，他它们相等；如果在长度或者内容上不等，它们不相等；两个字符串显示结果相同但是编码不同和=都认为他们不相等
7. 如果他们指向相同对象、数组、函数，它们相等；如果指向不同对象，他们不相等

==运算符判断相等的流程是怎样的

1. 如果两个值类型相同，按照===比较方法进行比较
2. 如果类型不同，使用如下规则进行比较
    1. 如果其中一个值是null，另一个是undefined，它们相等
    2. 如果一个值是**数字**另一个是**字符串**，将**字符串转换为数字**进行比较
    3. 如果有布尔类型，将**true转换为1，false转换为0**，然后用规则继续比较
    4. 如果一个值是对象，另一个是数字或字符串，将对象转换为原始值然后用规则继续比较
    5. **其他所有情况都认为不相等**

对象到字符串的转换步骤

1. 如果对象有toString()方法，javascript调用它。如果返回一个原始值（primitive value如：string number boolean）,将这个值转换为字符串作为结果
2. 如果对象没有toString()方法或者返回值不是原始值，javascript寻找对象的valueOf()方法，如果存在就调用它，返回结果是原始值则转为字符串作为结果
3. 否则，javascript不能从toString()或者valueOf()获得一个原始值，此时throws a TypeError

对象到数字的转换步骤

```
1. 1. 如果对象有valueOf()方法并且返回元素值，javascript将返回值转换为数字作为结果
```

2. 否则，如果对象有toString()并且返回原始值，javascript将返回结果转换为数字作为结果
3. 否则，throws a TypeError

<,>,<=,>=的比较规则

所有比较运算符都支持任意类型，但是**比较只支持数字和字符串**，所以需要执行必要的转换然后进行比较，转换规则如下：

1. 如果操作数是对象，转换为原始值：如果valueOf方法返回原始值，则使用这个值，否则使用toString方法的结果，如果转换失败则报错
2. 经过必要的对象到原始值的转换后，如果两个操作数都是字符串，按照字母顺序进行比较（他们的16位unicode值的大小）
3. 否则，如果有一个操作数不是字符串，**将两个操作数转换为数字**进行比较

+运算符工作流程

1. 如果有操作数是对象，转换为原始值
2. 此时如果有**一个操作数是字符串**，其他的操作数都转换为字符串并执行连接
3. 否则：**所有操作数都转换为数字并执行加法**

函数内部arguments变量有哪些特性,有哪些属性,如何将它转换为数组

- arguments所有函数中都包含的一个局部变量，是一个类数组对象，对应函数调用时的实参。如果函数定义同名参数会在调用时覆盖默认对象
- arguments[index]分别对应函数调用时的实参，并且通过arguments修改实参时会同时修改实参
- arguments.length为实参的个数（Function.length表示形参长度）
- arguments.callee为当前正在执行的函数本身，使用这个属性进行递归调用时需注意this的变化
- arguments.caller为调用当前函数的函数（已被遗弃）
- 转换为数组：`var args = Array.prototype.slice.call(arguments, 0);`

DOM事件模型是如何的,编写一个EventUtil工具类实现事件管理兼容

- DOM事件包含捕获（capture）和冒泡（bubble）两个阶段：捕获阶段事件从window开始触发事件然后通过祖先节点一次传递到触发事件的DOM元素上；冒泡阶段事件从初始元素依次向祖先节点传递直到window
- 标准事件监听elem.addEventListener(type, handler, capture)/elem.removeEventListener(type, handler, capture)：handler接收

保存事件信息的event对象作为参数，event.target为触发事件的对象，handler调用上下文this为绑定监听器的对象，event.preventDefault()取消事件默认行为，event.stopPropagation()/event.stopImmediatePropagation()取消事件传递

- 老版本IE事件监听elem.attachEvent('on'+type, handler)/elem.detachEvent('on'+type, handler)：handler不接收event作为参数，事件信息保存在window.event中，触发事件的对象为event.srcElement，handler执行上下文this为window使用闭包中调用handler.call(elem, event)可模仿标准模型，然后返回闭包，保证了监听器的移除。event.returnValue为false时取消事件默认行为，event.cancleBubble为true时取消时间传播
- 通常利用事件冒泡机制托管事件处理程序提高程序性能。

```
/**
 * 跨浏览器事件处理工具。只支持冒泡。不支持捕获
 * @author  (qiu_deqing@126.com)
 */

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, fals
e);
            return handler;
        } else if (elem.attachEvent) {
            var wrapper = function () {
              var event = window.event;
              event.target = event.srcElement;
              handler.call(elem, event);
            };
            elem.attachEvent('on' + type, wrapper);
            return wrapper;
        }
    },
    off: function (elem, type, handler) {
        if (elem.removeEventListener) {
            elem.removeEventListener(type, handler, fal
se);
        } 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;
        }
    },
    /**
     * keypress事件跨浏览器获取输入字符
     * 某些浏览器在一些特殊键上也触发keypress，此时返回null
     **/
     getChar: function (event) {
        if (event.which == null) {
            return String.fromCharCode(event.keyCode);
 // IE
        }
        else if (event.which != 0 && event.charCode !=
0) {
            return String.fromCharCode(event.which);
 // the rest
        }
        else {
            return null;    // special key
        }
     }
};
```

评价一下三种方法实现继承的优缺点,并改进

```
function Shape() {}

function Rect() {}

// 方法1
Rect.prototype = new Shape();

// 方法2
Rect.prototype = Shape.prototype;

// 方法3
Rect.prototype = Object.create(Shape.prototype);

Rect.prototype.area = function () {
  // do something
};
```

方法1:

1. 优点：正确设置原型链实现继承
2. 优点：父类实例属性得到继承，原型链查找效率提高，也能为一些属性提供合理的默认值
3. 缺点：父类实例属性为引用类型时，不恰当地修改会导致所有子类被修改
4. 缺点：创建父类实例作为子类原型时，可能无法确定构造函数需要的合理参数，这样提供的参数继承给子类没有实际意义，当子类需要这些参数时应该在构造函数中进行初始化和设置
5. 总结：继承应该是继承方法而不是属性，为子类设置父类实例属性应该是通过在子类构造函数中调用父类构造函数进行初始化

方法2:

1. 优点：正确设置原型链实现继承
2. 缺点：父类构造函数原型与子类相同。修改子类原型添加方法会修改父类

方法3:

1. 优点：正确设置原型链且避免方法1.2中的缺点
2. 缺点：ES5方法需要注意兼容性

改进：

1. 所有三种方法应该在子类构造函数中调用父类构造函数实现实例属性初始化

```
function Rect() {
    Shape.call(this);
}
```

2. 用新创建的对象替代子类默认原型，设置`Rect.prototype.constructor = Rect;`保证一致性
3. 第三种方法的polyfill：

```
function create(obj) {
    if (Object.create) {
        return Object.create(obj);
    }

    function f() {};
    f.prototype = obj;
    return new f();
}
```