عنوان: **آشنایی با مفاهیم شیء گرایی در جاوا اسکریپت #2**
نویسنده: مهران ر
تاریخ: ۱۳۹۲/۱۰/۰۸ ۱۸:۵
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, OOP

از آنجا که برای کار با جاوا اسکریپت نیاز به درک کاملی درباره‌ی مفهوم حوزه کارکرد متغیرها (Scope) می‌باشد و نحوه فراخوانی توابع نیز نقش اساسی در این مورد بازی می‌کند، در این قسمت با این موارد آشنا خواهیم شد:
جاوا اسکریپت از مفهومی به نام functional scope برای تعیین حوزه متغیرها استفاده می‌کند و به این معنی است که با تعریف توابع، حوزه عملکرد متغیر مشخص می‌شود. در واقع هر متغیری که در یک تابع تعریف می‌شود در کلیه قسمتهای آن تابع، از قبیل If statement – for loops و حتی nested function نیز در دسترس میباشد.
اجازه دهید با مثالی این موضوع را بررسی نماییم.

```
function testScope() {
var myTest = true;
if (true) {
var myTest = "I am changed!"
}
alert(myTest);
}
testScope(); // will alert "I am changed!"
```

همانگونه که میبینیم با اینکه در داخل بلاک if یک متغیر جدید تعریف شده، ولی در خارج از این بلاک نیز این متغیر قابل دسترسی میباشد. البته در مثال بالا اگر بخواهیم به متغیر myTest در خارج از function دسترسی داشته باشیم، با خطای undefined مواجه خواهیم شد. یعنی برای مثال در کد زیر:

```
function testScope() {
var myTest = true;
if (true) {
var myTest = "I am changed!"
}
alert(myTest);
}
testScope(); // will alert "I am changed!"
alert(myTest); // will throw a reference error, because it doesn't exist outside of the function
```

برای حل این مشکل دو راه وجود دارد:

1 – متغیر myTest را در بیرون بلاک testScope() تعریف کنیم
2 – هنگام تعریف متغیر myTest، کلمه کلیدی var را حذف کنیم که این موضوع باعث میشود این متغیر در کل window قابل دسترس باشد و یا به عبارتی متغیر global میشود.
قبل از پرداختن به ادامه بحث خواندن مقاله مربوط به Closure در جاوااسکریپت توصیه میگردد .
در پایان بحث Scopeها با یک مثال نسبتا جامع اکثر این حالات به همراه خروجی را نشان میدهیم :

```
<script type="text/javascript">
          // a globally-scoped variable
        var a = 1;
        // global scope
        function one()
        {
            alert(a);
        }
        // local scope
        function two(a)
        {
            alert(a);
        }
        // local scope again
        function three()
        {
            var a = 3;
```

```
            alert(a);
        }
        // Intermediate: no such thing as block scope in javascript
        function four()
        {
            if (true)
            {
                var a = 4;
            }
            alert(a); // alerts '4', not the global value of '1'
        }
        // Intermediate: object properties
        function Five()
        {
            this.a = 5;
        }
        // Advanced: closure
        var six = function ()
        {
            var foo = 6;
            return function ()
            {
                // javascript "closure" means I have access to foo in here,
                // because it is defined in the function in which I was defined.
                alert(foo);
            }
        }()
        // Advanced: prototype-based scope resolution
        function Seven()
        {
            this.a = 7;
        }
        // [object].prototype.property loses to [object].property in the lookup chain
        Seven.prototype.a = -1; // won't get reached, because 'a' is set in the constructor above.
        Seven.prototype.b = 8; // Will get reached, even though 'b' is NOT set in the constructor.
        // These will print 1-8
        one();
        two(2);
        three();
        four();
        alert(new Five().a);
        six();
        alert(new Seven().a);
        alert(new Seven().b);
</Script>
```

برای مطالعه بیشتر به اینجا مراجعه نمایید.

Function Invocation Patterns In JavaScript :

از آنجا که توابع در جاوااسکریپت به منظور 1 – ساخت اشیاء و 2 – حوزه دسترسی متغیرها(Scope) نقش اساسی ایفا می‌کنند بهتر است کمی درباره استفاده و نحوه فراخوانی آنها (Function Invocation Patterns) در جاوااسکریپت بحث نماییم.

در جاوااسکریپت 4 مدل فراخوانی تابع داریم که به نامهای زیر مطرح هستند:

1. Method Invocation
2. Function Invocation
3. Constructor Invocation
4. Apply And Call Invocation

در فراخوانی توابع به هر یک از روشهای بالا باید به این نکته توجه داشت که حوزه دسترسی متغیرها در جاوااسکریپت ابتدا و انتهای توابع هستند و اگر به عنوان مثال از توابع تو در تو استفاده کردیم ،حوزه شی this برای توابع داخلی تغییر خواهد کرد .این موضوع را در طی مثالهایی نشان خواهیم داد. **Method Invocation :**

وقتی یک تابع قسمتی از یک شی باشد به آن متد میگوییم به عنوان مثال :

```
var obj = {
    value: 0,
    increment: function() {
        this.value+=1;
    }
};
obj.increment(); //Method invocation
```

در اینحالت this به شی (Object) اشاره میکند که متد در آن فراخوانی شده است و در زمان اجرا نیز به عناصر شی Bind میشود ،در مثال بالا حوزه this شی obj خواهد شد و به همین منظور به متغیر value دسترسی داریم. **Function Invocation:**
در اینحالت که از () برای فراخوانی تابع استفاده میگردد ،This به شی سراسری (global object) اشاره می‌کند؛ منظور اینکه this به اجزای تابعی که فراخوانی آن انجام شده اشاره نمی‌کند. اجازه دهید با مثالی این موضوع را روشن کنیم

```
<script type="text/javascript">
var value = 500; //Global variable
var obj = {
    value: 0,
    increment: function() {
        this.value++;
        var innerFunction = function() {
            alert(this.value);
        }
        innerFunction(); //Function invocation pattern
    }
}
obj.increment(); //Method invocation pattern
<script type="text/javascript">
Result : 500
```

از آنجا که () innerFunction به شکل Function invocation pattern فراخوانی شده است به متغیر value در داخل تابع increment دسترسی نداریم و حوزه دسترسی global میشود و اگر در حوزه global نیز این متغیر تعریف نشده بود به خطای undefined میرسیدیم .
برای حل این گونه مشکلات ساختار کد نویسی ما بایستی به شکل زیر باشد :

```
<script type="text/javascript">
var value = 500; //Global variable
var obj = {
    value: 0,
    increment: function() {
        var that = this;
        that.value++;
        var innerFunction = function() {
            alert(that.value);
        }
        innerFunction(); //Function invocation pattern
    }
}
obj.increment();
<script type="text/javascript">
Result : 1
```

در واقع با تعریف یک متغیر با نام مثلا that و انتساب شی this به آن میتوان در توابع بعدی که به شکل Function invocation pattern فراخوانی میگردند به این متغیر دسترسی داشت . **Constructor Invocation :**
در این روش برای فراخوانی تابع از کلمه new استفاده میکنیم. در این حالت یک شیء مجزا ایجاد شده و به متغیر دلخواه ما اختصاص پیدا می‌کند. به عنوان مثال داریم :

```
var Dog = function(name) {
  //this == brand new object ({});
    this.name = name;
    this.age = (Math.random() * 5) + 1;
};
var myDog = new Dog('Spike');
//myDog.name == 'Spike'
//myDog.age == 2
var yourDog = new Dog('Spot');
//yourDog.name == 'Spot'
//yourDog.age == 4
```

در این مورد با استفاده از New باعث میشویم همه خواص و متدهای تابع function برای هر نمونه از آن که ساخته میشود (از طریق مفهوم Prototype که قبلا درباره آن بحث شد) بطور مجزا اختصاص یابد. در مثال بالا شی mydog چون حاوی یک نمونه از

تابع dog بصورت Constructor Invocation میباشد، در نتیجه به خواص تابع dog از قبیل name و age دسترسی داریم. در اینجا اگر کلمه new استفاده نشود به این خواص دسترسی نداریم؛ در واقع با اینکار، this به mydog اختصاص پیدا میکند.
اگر از new استفاده نشود متغیر undefined، myDog میشود.
یک مثال دیگر :

```
var createCallBack = function(init) { //First function
    return new function() { //Second function by Constructor Invocation
        var that = this;
        this.message = init;
        return function() { //Third function
            alert(that.message);
        }
    }
}
window.addEventListener('load', createCallBack("First Message"));
window.addEventListener('load', createCallBack("Second Message"));
```

در مثال بالا از مفهوم closure نیز در مثالمان استفاده کرده ایم . **Apply And Call Invocation:**

تمامی توابع جاوااسکریپت دارای دو متد توکار apply() و call() هستند که توسط این متدها میتوان این توابع را با context دلخواه فراخوانی کرد.
نحوه فراخوانی به شکل مقابل است :

```
myFunction.apply(thisContext, arrArgs);
myFunction.call(thisContext, arg1, arg2, arg3, ..., argN);
```

که thisContext به حوزه اجرایی (execution context) تابع اشاره میکند. تفاوت دو متد apply() و call() در نحوه فرستادن آرگومانها به تابع میباشد که در اولی توسط آرایه اینکار انجام میشود و در دومی همه آرگومانها را بطور صریح نوشته و با کاما از هم جدا میکنیم .

مثال :

```
var contextObject = {
testContext: 10
}
var otherContextObject = {
testContext: "Hello World!"
}
var testContext = 15; // Global variable
function testFunction() {
alert(this.testContext);
}
testFunction(); // This will alert 15
testFunction.call(contextObject); // Will alert 10
testFunction.apply(otherContextObject); // Will alert "Hello World”
```

در این مثال دو شی متفاوت با خواص همنام تعریف کرده و یک متغیر global نیز تعریف میکنیم. در انتها یک تابع تعریف میکنیم که مقدار this.testContext را نمایش میدهد. در ابتدا حوزه اجرایی تابع (this) کل window جاری میباشد و وقتی testFunction() اجرا شود مقدار متغیر global نمایش داده میشود. در اجرای دوم this به contextObject اشاره کرده و حوزه اجرایی عوض میشود و در نتیجه مقدار testContext مربوطه که در این حالت 10 میباشد نمایش داده میشود و برای فراخوانی سوم نیز به همین شکل .

یک مثال کاملتر :

```
var o = {
  i : 0,
  F : function() {
    var a = function() { this.i = 42; };
    a();
```

```
    document.write(this.i);
  }
};
o.F();
Result :0
```

خط o.f() تابع f را به شکل Method invocation اجرا میکند. در داخل تابع f یک تابع دیگر به شکل function invocation اجرا میشود که در اینحال this به global object اشاره میکند و باعث میشود مقدار i در خروجی 0 چاپ شود .
برای حل این مشکل 2 راه وجود دارد
راه اول :

```
var p = {
  i : 0,
  F : function() {
    var a = function() { this.i = 42; };
    a.apply(this);
    document.write(this.i);
  }
};
 p.F();
Result :42
```

با اینکار this را موقع اجرای تابع درونی برایش فرستاده تا حوزه اجرای تابع عوض شود و به i دسترسی پیدا کنیم .
یا اینکه همانند مثالهای قبلی :

```
var q = {
  i: 0,
  F: function F() {
    var that = this;
    var a = function () {
      that.i = 42;
    }
    a();
    document.write(this.i);
  }
}
 q.F();
```

منابع :

Javascript programmer,s refrence http://coding.abel.nu/2013/03/more-on-this-in-javascript/
http://seanmonstar.com/post/707068021/4-ways-functions-mess-with-this