عنوان: **آشنایی با مفاهیم شیء گرایی در جاوا اسکریپت #1**
نویسنده: مهران ر
تاریخ: ۱۳۹۲/۱۰/۰۴ ۱۴:۳۰
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, OOP

با توجه به فراگیر شدن استفاده از جاوا اسکریپت و بخصوص مبحث شیء گرایی، تصمیم گرفتم طی سلسله مقالاتی با مباحث شیء گرایی در این زبان بیشتر آشنا شویم. جاوا اسکریپت یک زبان مبتنی بر شیء است و نه شیءگرا و خصوصیات زبان‌های شیء گرا، به طور کامل در آن پیاده سازی نمی‌گردد.
لازم به ذکر است که انواع داده‌ای در جاوا اسکریپت شامل 2 نوع می‌باشند:
-1 نوع داده اولیه (Primitive) که شامل Number ، Boolean و Strings می‌باشند.
-2 نوع داده Object که طبق تعریف هر Object مجموعه‌ای از خواص و متدها است.
نوع داده‌ای اولیه، از نوع Value Type و نوع داده ای Object، از نوع Refrence Type می‌باشد.

برای تعریف یک شیء (Object) در جاوا اسکریپت، 3 راه وجود دارد:
1 - تعریف و ایجاد یک نمونه مستقیم از یک شیء (direct instance of an object)
2 – استفاده از function برای تعریف و سپس نمونه سازی از یک شیء (Object Constructor)
3 – استفاده از متد Object.Create

روش اول :
در روش اول دو راه برای ایجاد اشیاء استفاده می‌گردد که با استفاده از دو مثال ذیل، این دو روش توضیح داده شده‌اند:

مثال اول : (استفاده از new)

```
<script type=”text/javascript”>
 var person = new Object();
  person.firstname = “John”;
  person.lastname = “Doe”;
  person.age = 50;
  person.eyecolor = “blue”;
  document.write(person.firstname + “ is “ + person.age + “ years old.”);
</script>
```

```
result : John is 50 years old.
```

در این مورد، ابتدا یک شیء پایه ایجاد می‌گردد و خواص مورد نظر برایش تعریف می‌گردد و با استفاده از اسم شیء به این خواص دسترسی داریم.

مثال دوم (استفاده از literal notation)

```
<script type=”text/javascript”>
var obj = {
   var1: “text1”,
   var2: 5,
   Method: function ()
   {
     alert(this.var1);
   }
  };
  obj.Method();
</script>
```

```
Result : text1
```

در این مورد با استفاده از کلمه کلیدی var یک شیء تعریف می‌شود و در داخل {} کلیه خواص و متدهای این شیء تعریف می‌گردد. این روش برای تعریف اشیاء در جاوا اسکریپت بسیار متداول است.
هر دو مثالهای 1 و 2 در روش اول برای ایجاد اشیاء بکار میروند. امکان گسترش دادن اشیاء در این روش و اضافه کردن خواص و متد در آینده نیز وجود دارد. بعنوان مثال می‌توان نوشت :

```
Obj.var3 = “text3”;
```

در این‌حال، خاصیت سومی به مجموع خواص شی Obj اضافه می‌گردد.
حال در این مثال اگر مقدار شی obj را برابر یک شیء دیگر قرار دهیم به نحو زیر :

```
var newobj = obj;
  newobj.var1 = "other text";
  alert(obj.var1);// other text
  alert(newobj.var1);// other text
```

و برای اینکه بتوان از امکانات زبانهای شیء گرا در این زبان استفاده کرد، بایستی الگویی را تعریف کنیم و سپس از روی این الگو، اشیا مورد نظر را پیاده سازی نمائیم.
می‌بینیم که مقدار هر دو متغیر در خروجی یکسان می‌باشد و این موضوع با ماهیت شیء گرایی که در آن همه‌ی اشیایی که از روی یک الگو نمونه سازی می‌گردند مشخصه‌هایی یکسان، ولی مقادیر متفاوتی دارند، متفاوت است. البته این موضوع از آنجا ناشی می‌گردد که اشیاء ایجاد شده در جاوا اسکریپت ذاتا type refrence هستند و به همین منظور برای پیاده سازی الگویی (کلاسی) که بتوان رفتار شیء گرایی را از آن انتظار داشت از روش زیر استفاده میکنیم. برای درک بهتر اسم این الگو را کلاس مینامیم که در روش دوم به آن اشاره میکنیم.

روش دوم :

```
<script type=”text/javascript”>
  function Person(firstname, lastname, age, eyecolor)
  {
     this.firstname = firstname;
     this.lastname = lastname;
     this.age = age;
     this.eyecolor = eyecolor;
  }

   var myFather = new Person("John", "Doe", 50, "blue");
  document.write(myFather.firstname + " is " + myFather.age + " years old.");
  result : John is 50 years old.

   var myMother=new person("Sally","Rally",48,"green");
   document.write(myMother.firstname + " is " + myFather.age + " years old.");
   result : Sally is 48 years old.
  </script>
```

یا به شکل زیر :

```
var Person = function (firstname, lastname, age, eyecolor)
  {
     this.firstname = firstname;
     this.lastname = lastname;
     this.age = age;
     this.eyecolor = eyecolor;
  }

  var myFather = new Person("John", "Doe", 50, "blue");
  document.write(myFather.firstname + " is " + myFather.age + " years old.");
   result : John is 50 years old.
```

```
   var myMother=new person("Sally","Rally",48,"green");
   document.write(myMother.firstname + " is " + myFather.age + " years old.");
   result : Sally is 48 years old.
```

به این روش Object Constructor یا سازنده اشیاء گفته می‌شود.
در اینجا با استفاده از کلمه کلیدی function و در داخل {} کلیه خواص و متدهای لازم را به شیء مورد نظر اضافه می‌کنیم.
استفاده از کلمه this در داخل function به این معنی است که هر کدام از نمونه‌های object مورد نظر، مقادیر متفاوتی خواهند داشت .

یک مثال دیگر :

```
<script type="text/javascript">
   function cat(name) {
      this.name = name;
      this.talk = function() {
      alert( this.name + " say meeow!" )
   }
 }

cat1 = new cat("felix")
cat1.talk() //alerts "felix says meeow!"
cat2 = new cat("ginger")
cat2.talk() //alerts "ginger says meeow!"
</Script>
```

در اینجا می‌بینیم که به ازای هر نمونه از اشیایی که با function می‌سازیم، خروجی متفاوتی تولید می‌گردد که همان ماهیت شیء گرایی است.

روش سوم :استفاده از متد Object.Create

```
var myObjectLiteral = {
    property1: "one",
    property2: "two",
    method1: function() {
       alert("Hello world!");
}}
var myChild = Object.create(myObjectLiteral);
myChild.method1(); // will alert "Hello world!"
```

در این روش با استفاده از متد Object.Create و استفاده از یک شیء که از قبل ایجاد شده، یک شیء جدید ایجاد می‌شود.
حال برای اضافه کردن متدها و خاصیت‌هایی به کلاس جاوا اسکریپتی مورد نظر، به طوریکه همه‌ی نمونه‌هایی که از این کلاس ایجاد می‌شوند بتوانند به این متدها و خاصیت‌ها دسترسی داشته باشند، از مفهومی به اسم prototype استفاده می‌کنیم. برای مثال کلاس زیر را در نظر بگیرید:
این کلاس یک سیستم ساده امتحانی (quiz) را پیاده می‌کند که در آن اطلاعات شخص که شامل نام و ایمیل می‌باشد گرفته شده و سه تابع، شامل ذخیره نمرات، تغییر ایمیل و نمایش اطلاعات شخص به همراه نمرات نیز به آن اضافه می‌شود.

```
function User (theName, theEmail) {
    this.name = theName;
    this.email = theEmail;
    this.quizScores = [];
    this.currentScore = 0;
}
```

حال برای اضافه نمودن متدهای مختلف به این کلاس داریم :

```
User.prototype = {
  saveScore:function (theScoreToAdd)  {
   this.quizScores.push(theScoreToAdd)
  },
  showNameAndScores:function ()  {
    var scores = this.quizScores.length > 0 ? this.quizScores.join(",") : "No Scores Yet";
    return this.name + " Scores: " + scores;
  },
  changeEmail:function (newEmail)  {
    this.email = newEmail;
    return "New Email Saved: " + this.email;
  }
}
```

و سپس برای استفاده از آن و گرفتن خروجی نمونه داریم :

```
// A User
  firstUser = new User("Richard", "Richard@examnple.com");
  firstUser.changeEmail("RichardB@examnple.com");
  firstUser.saveScore(15);
  firstUser.saveScore(10);
  document.write(firstUser.showNameAndScores()); //Richard Scores: 15,10
  document.write('<br/>');
  // Another User
  secondUser = new User("Peter", "Peter@examnple.com");
  secondUser.saveScore(18);
  document.write(secondUser.showNameAndScores()); //Peter Scores: 18
```

در نتیجه تمام نمونه‌های کلاس User می‌توانند به این متدها دسترسی داشته باشند و به این صورت مفهوم Encapsulation نیز پیاده می‌گردد.

وراثت (Inheritance) در جاوا اسکریپت :

در بسیاری از مواقع لازم است عملکردی (Functionality) که در یک کلاس تعریف می‌گردد، در کلاسهای دیگر نیز در دسترس باشد. بدین منظور از مفهوم وراثت استفاده می‌شود. در نتیجه کلاس‌ها می‌توانند از توابع خود و همچنین توابعی که کلاسهای والد در اختیار آنها می‌گذارند استفاده کنند. برای این منظور چندین راه حل توسط توسعه دهندگان ایجاد شده است که در ادامه به چند نمونه از آنها اشاره می‌کنیم.
ساده‌ترین حالت ممکن از الگویی شبیه زیر است:

```
<script type="text/javascript">
function Base()
  {
     this.color = "blue";
  }
  function Sub()
  {
  }
  Sub.prototype = new Base();
  Sub.prototype.showColor = function ()
  {
     alert(this.color);
  }
  var instance = new Sub();
  instance.showColor(); //"blue"
</Script>
```

در کد بالا ابتدا یک class) function) به نام Base که حاوی یک خصوصیت به نام color می‌باشد، تعریف شده و سپس یک کلاس دیگر بنام sub تعریف می‌کنیم که قرار است خصوصیات و متدهای کلاس Base را به ارث ببرد و سپس از طریق خصوصیت prototype کلاس Sub، که نمونه‌ای از کلاس Base را به آن نسبت می‌دهیم باعث می‌شود خواص و متدهای کلاس Base توسط کلاس Sub قابل دسترسی باشد. در ادامه متد showColor را به کلاس Sub اضافه می‌کنیم و توسط آن به خصوصیت color در این کلاس دسترسی پیدا میکنیم.

راه حل دیگری نیز برای اینکار وجود دارد که الگویی است بنام Parasitic Combination :
در این الگو براحتی و با استفاده از متد Object.create که در بالا توضیح داده شد، هر کلاسی که ایجاد میکنیم، با انتساب آن به یک شیء جدید، کلیه خواص و متدهای آن نیز توسط شیء جدید قابل استفاده میشود.

```
<script language="javascript" type="text/javascript">
if (typeof Object.create !== 'function') {
Object.create = function (o) {
//ایجاد یک کلاس خالی که قرار است خواص کلاس دریافتی توسط آرگومان کلاس پایه را به ارث ببرد
  function F() {
}
با انتساب F  توسط خصوصیت  Prototype باعث میشویم کلیه خواص و متدهای دریافتی توسط  F با ارث برده شود
آرگومان دریافتی که یک شی است به کلاس
  F.prototype = o;
   return new F();
  };
}

var cars = {
type: "sedan",
wheels: 4
  };
  // We want to inherit from the cars object, so we do:
  var toyota = Object.create(cars);
// now toyota inherits the properties from cars
  document.write(toyota.type);
</script>
output :sedan
```

در قسمتهای دیگر به مباحثی همچون Override و CallBaseMethodها خواهیم پرداخت.

برای مطالعه بیشتر :

http://eloquentjavascript.net/chapter8.html

http://phrogz.net/JS/classes/OOPinJS2.html