عنوان: **آشنایی با Promises در جاوا اسکریپت**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۱۰/۱۲ ۸:۰
آدرس: www.dotnettips.info
گروه‌ها: JavaScript, jQuery Ajax, AngularJS

در حین انجام اعمال غیرهمزمان جاوا اسکریپتی مانند فراخوانی‌های jQuery AJAX، برای مدیریت دریافت نتایج، عموما از یک سری callback استفاده می‌شود. برای مثال:

```
$.get('http://site-url', function(data) {
//این تابع پس از پایان کار عملیات ای‌جکسی در آینده فراخوانی خواهد شد
});
```

تا اینجا مشکلی به نظر نمی‌رسد. اما مورد ذیل چطور؟

```
$.get('http://site-url/0', function(data0) {
    // callback #1
    $.get('http://site-url/1', function(data1) {
        // callback #2
        $.post('http://site-url/2', function(data2) {
            // callback #3
        });
    });
});
```

در اینجا نیاز است پس از پایان کار عملیات Ajax ایی اول، عملیات دوم و پس از آن عملیات سومی انجام شود. همانطور که مشاهده می‌کنید، این نوع کدها به سرعت از کنترل خارج می‌شوند؛ خوانایی پایینی داشته و مدیریت استثناءهای رخ داده در آن‌ها نیز در این بین مشکل است. از این جهت که خطاهای هر کدام به سطحی بالاتر منتقل نمی‌شود و باید همانجا محلی و داخل هر callback مدیریت گردد.
روش‌های زیادی برای حل این مساله ارائه شده‌است و در حال حاضر کار کردن با promiseها متداول‌ترین روش حل مدیریت فراخوانی کدهای همزمان جاوا اسکریپتی است. برای نمونه اگر از AngularJS استفاده کنید، سرویس‌های آن برای دریافت اطلاعات از سرور، از یک چنین مفهومی استفاده می‌کنند.

Promise در جاوا اسکریپت چیست؟

شیء Promise، نمایانگر قراردادی است که در آینده می‌تواند مورد قبول واقع شود، یا رد گردد. بررسی این قرارداد، تنها یکبار می‌تواند رخ دهد (پذیرش یا رد آن). هنگامیکه این بررسی صورت گرفت (رد یا پذیرش آن و نه هردو)، یک callback برای اطلاع رسانی فراخوانی می‌گردد. سپس این callback می‌تواند یک Promise دیگر را سبب شود. به این ترتیب می‌توان Promiseها را زنجیر وار به یکدیگر متصل کرد. برای نمونه jQuery به صورت توکار از promises پشتیبانی می‌کند:

```
// returns a promise
$.get('http://site-url/0')
.then(function(data) {
    // callback 1
    // returns a promise
    return $.get('http://site-url/1');
})
.then(function(data) {
    // callback 2
    // returns a promise
    return $.post('http://site-url/2');
})
.then(function(data) {
    // callback 3
});
```

متد get در jQuery یک شیء promise را بازگشت می‌دهد. در ادامه می‌توان این نتیجه را توسط متد then، زنجیروار ادامه داد. متدی که به عنوان پارامتر به then ارسال می‌شود، یک callback بوده و پس از پایان کار promise قبلی رخ می‌دهد. آرگومانی که به

این callback ارسال می‌شود، نتیجه‌ی promise قبلی است. در حین اعمال jQuery Ajax، این callback تنها زمانی فراخونی می‌شود که عملیات قبلی موفقیت آمیز بوده باشد و data ارائه شده، اطلاعاتی است که توسط response دریافتی از سرور، دریافت گردیده‌است.

در این حالت، هر callback حداقل سه کار را می‌تواند انجام دهد:

الف) یک promise دیگر را بازگشت دهد. نمونه آن‌را با return $.get در کدهای فوق ملاحظه می‌کنید.

ب) خاتمه عادی. همینجا کار promise با مقدار بازگشت داده شده، پایان می‌یابد.

ج) صدور یک استثناء. سبب برگشت خوردن و عدم پذیرش promise می‌شود.

استفاده از Promises در سایر کتابخانه‌ها

jQuery پیاده سازی توکاری از promises دارد؛ اما سایر کتابخانه‌ها، مانند AngularJS ایی که مثال زده شده چطور عمل می‌کنند؟ استانداردی به نام Promises/A+ جهت یک دست سازی پیاده سازی‌های promise در جاوا اسکریپت پیشنهاد شده‌است. jQuery نیمی از آن‌را پیاده سازی کرده‌است؛ اما کتابخانه‌ی دیگری به نام Q Library ، پیاده سازی نسبتا مفصل‌تری را از این استاندارد ارائه می‌دهد. فریم ورک AngularJS نیز در پشت صحنه از همین کتابخانه برای پیاده سازی promises استفاده می‌کند.

آشنایی با کتابخانه Q

استفاده مقدماتی از Q همانند مثالی است که از jQuery ملاحظه کردید.

```
Q.fcall(callback1)
.then(callback2);
```

اشیاء promise بازگشت داده شده توسط jQuery نیز توسط کتابخانه Q مورد پذیرش واقع می‌شوند:

```
Q.fcall(function() {
    return $.get('http://my-url');
})
.then(callback3);
```

علاوه بر این‌ها مفهومی به نام deferred objects نیز در کتابخانه‌ی Q پیاده سازی شده‌است:

```
function waitForClick() {
    var deferred = Q.defer();

$('#okButton').click(function() {
        deferred.resolve();
    });

$('#cancelButton').click(function() {
        deferred.reject();
    });

return deferred.promise;
}

Q.fcall(waitForClick)
.then(function() {
    //  ok button was clicked
}, function() {
    //  cancel button was clicked
});
```

توسط deferred objects می‌توان بررسی یک promise را به تاخیر انداخت. در مثال فوق، اولین callback فراخوانی شده به نام waitForClick، از اشیاء به تاخیر افتاده استفاده می‌کند. ابتدا توسط فراخوانی متد Q.defer، یک deferred object ایجاد می‌شود. در این بین اگر کاربر بر روی دکمه‌ی OK کلیک کرد، با فراخوانی deferred.resolve، این promise مورد پذیرش واقع خواهد شد و یا اگر کاربر بر روی دکمه‌ی cancel کلیک کند، با فراخوانی متد deferred.reject، این promise رد می‌گردد. نهایتا شیء promise

توسط deferred.promise بازگشت داده خواهد شد.
در ادامه کار، اینبار متد then، دو callback را قبول می‌کند. Callback اول پس از پذیرش قرار داد و Callback دوم پس از رد قرار داد، فراخوانی خواهد گردید.
در رنجیره تعریف شده، اگر معادلی برای reject درنظر گرفته نشده باشد، مانند مثال ذیل:

```
Q.fcall(myFunction1)
.then(success1)
.then(success2, failure1);
```

Q به دنبال نزدیک‌ترین متد callback گزارش خطای کار خواهد گشت. در این حالت متد failure1 در صورت شکست اولین promise فراخوانی خواهد شد.
همچنین اگر نتیجه‌ی success1 با شکست مواجه شود نیز failure1 فراخوانی می‌گردد. اما باید درنظر داشت که شکست success2، توسط failure1 مدیریت نمی‌شود.

Promises در AngularJS

در AngularJS امکانات کتابخانه Q توسط پارامتری به نام $q در اختیار سرویس‌های برنامه قرار می‌گیرد (تزریق می‌شود):

```
var app = angular.module("myApp", []);
app.factory('dataSvc', function($http, $q){
  var basePath="api/books";
  getAllBooks = function(){
   var deferred = $q.defer();
 $http.get(basePath).success(function(data){
    deferred.resolve(data);
   }).error(function(err){
    deferred.reject("service failed!");
   });
   return deferred.promise;
  };

  return{
   getAllBooks:getAllBooks
  };
});

app.controller('HomeController', function($scope, $window, dataSvc){
 function initialize(){
   dataSvc.getAllBooks().then(function(data){
    $scope.books = data;
   }, function(msg){
    $window.alert(msg);
   });
 }

 initialize();
});
```

در اینجا اگر دقت کنید، مباحث و عملکرد آن دقیقا مانند قبل است. ابتدا یک deferred object با فراخوانی متد q.defer ایجاد شده است. سپس با استفاده از امکانات توکار http آن (بجای استفاده از jQuery Ajax)، کار فراخوانی یک restful service صورت گرفته است (مثلا فراخوانی یک ASP.NET Web API). در صورت موفقیت کار، متد deferred.resolve و در صورت عدم موفقیت، متد deferred.reject فراخوانی شده‌است. نهایتا این سرویس، یک deferred.promise را بازگشت می‌دهد.
اکنون در کنترلری که قرار است از این سرویس استفاده کند، متد then کتابخانه Q را ملاحظه می‌کنید که دو Callback متناظر resolve و reject مدیریت promise بازگشت داده شده را به همراه دارد. اگر عملیات Ajaxایی موفقیت آمیز باشد، شیء books را مقدار دهی می‌کند و اگر خیر، پیامی را به کاربر نمایش خواهد داد.

پشتیبانی مرورگرهای جدید از استاندارد Promise

در حال حاضر کروم 32 و نگارش‌های شبانه فایرفاکس، Promise را که جزئی از استاندارد JavaScript شده‌است، به صورت توکار

و بدون نیاز به کتابخانه‌های جانبی، پشتیبانی می‌کنند.

```
if (window.Promise) { // Check if the browser supports Promises
 var promise = new Promise(function(resolve, reject) {
  //asynchronous code goes here
 });
}
```

در اینجا با فراخوانی window.Promise مشخص می‌شود که آیا مرورگر جاری از Promises پشتیبانی می‌کند یا خیر. سپس یک شیء promise ایجاد شده و این شیء توسط پارامترهای resolve و reject که هر دو تابع می‌باشند، کار مدیریت کدهای غیرهمزمان را انجام می‌دهد:

```
if (window.Promise) {
 console.log('Promise found');

 var promise = new Promise(function(resolve, reject) {
      // async
  if (result) {
   resolve(data);
  } else {
   reject('error');
  }
 });

 promise.then(function(data) {
  console.log('Promise fulfilled.');
 }, function(error) {
  console.log('Promise rejected.');
 });
} else {
 console.log('Promise not available');
}
```

در مثال فوق ابتدا یک شیء Promise ایجاد شده است. این شیء استاندارد بوده و با کروم 32 قابل آزمایش است. سپس در callback ابتدایی آن می‌توان یک عملیات AJAX ایی را انجام داد. اگر نتیجه‌ی آن موفقیت آمیز بود، تنها کافی است پارامتر اول این callback را فراخوانی کنیم و اگر خیر، پارامتر دوم آن‌را. برای استفاده از این شیء Promise ایجاد شده، می‌توان از متد then استفاده کرد. این متد نیز در اینجا دو callback پذیرش و رد promise را می‌تواند دریافت کند. برای زنجیر کردن آن کافی است متد then، یک Promise دیگر را بازگشت دهد و از نتیجه‌ی آن در then بعدی استفاده گردد.