عنوان: **RequireJs**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۶/۳۰ ۸:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: requirejs, module

در طراحی و توسعه پروژه‌های تحت وب در مقیاس بزرگ برای اینکه مدیریت پروژه راحت‌تر شود کدهای مورد نظر را در چند ماژول قرار می‌دهند در نتیجه کدهای پروژه در بلاک‌های کوچک‌تر قرار خواهند داشت. نوشتن پروژه به صورت ماژولار قابلیت استفاده مجدد از کدهای برنامه را افزایش می‌دهد، علاوه بر آن مدیریت پروژه در فاز نگهداری آسان‌تر خواهد شد؛ از طرفی دیگر وابستگی بین ماژول‌ها و تامین آن ها، همواره مهم‌ترین مفهوم برای توسعه دهندگان پروژه‌های وب است. RequireJs یکی از فریم ورک‌های محبوب برای مدیریت وابستگی‌های بین ماژول‌ها است و کاربرد اصلی آن راحت سازی مفهوم modularity در اینگونه پروژه هاست.
پروژه‌های بزرگ عموما دارای یک یا چند فایل جاوااسکریپ هستند که برای استفاده از آن‌ها در صفحات از تگ script استفاده می‌شود. اگر این فایل‌ها دارای وابستگی به هم باشند، ترتیب فراخوانی این فایل در تگ script مهم است. برای مثال:
یک پروژه دارای فایل‌های زیر خواهد بود:

purchase.js

```
function purchaseProduct(){
  console.log("Function : purchaseProduct");
  var credits = getCredits();
  if(credits > 0){
    reserveProduct();
    return true;
  }
  return false;
}
```

product.js

```
function reserveProduct(){
  console.log("Function : reserveProduct");
  return true;
}
```

credits.js

```
function getCredits(){
  console.log("Function : getCredits");
  var credits = "100";
  return credits;
}
```

همان طور که در فایل‌های بالا مشاهده می‌کنید در فایل purchase.js از دو فایل دیگر استفاده شده است. در فایل main.js پروژه کد زیر را برای استفاده از فایل‌های بالا می‌نویسیم:

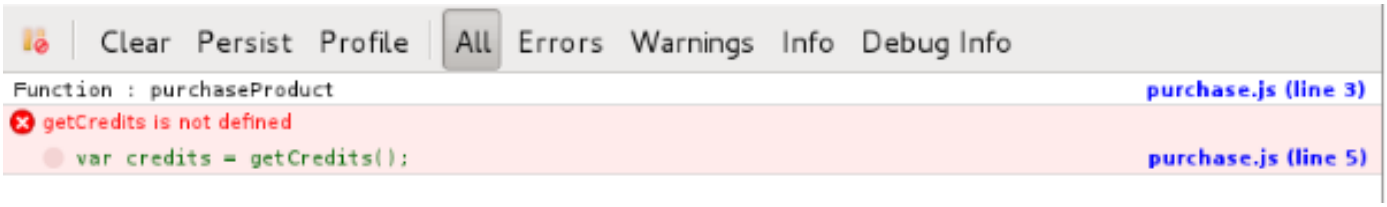
main.js

```
var result = purchaseProduct();
```

فرض کنید در فایل Html تگ‌های script ما به صورت زیر باشد:

```
<script src="products.js"></script>
<script src="purchase.js"></script>
<script src="main.js"></script>
<script src="credits.js"></script>
```

از آنجا که لود فایل purchase.js وابستگی مستقیم به لود فایل credits.js دارد و از طرفی دیگر به دلیل این که فایل credit.Js بعد از تمام فایل‌ها لود می‌شود در نتیجه برنامه اجرا نخواهد شد و با خطای زیر روبرو می‌شویم:



فقط کافیست تصور کنید که تعداد و حجم کدهای این فایل‌ها در یک پروژه زیاد باشد یا حتی این فایل‌ها توسط یک برنامه نویس دیگر تهیه و تدوین شده باشد؛ در نتیجه به خاطر سپردن این وابستگی‌ها به طور قطع کار سخت خواهد بود و در خیلی موارد طاقت فرسا.

RequireJs چگونه برای حل این مشکل به ما کمک می‌کند؟
با استفاده از فریم ورک RequireJs کدهای ما به ماژول‌های کوچک‌تر شکسته می‌شود و وابستگی ماژول‌ها در تنظیمات لود فایل ثبت می‌شود. در ضمن این فریم ورک با مرورگرها جدید و محبوب کاملا سازگار است. برای شروع فایل‌های مورد نیاز را از اینجا دانلود نمایید. البته می‌توانید از nuget هم استفاده کنید:

```
PM> Install-Package RequireJS
```

در مثال بالا فایل‌های جاوااسکریپ به صورت زیر تغییر خواهد کرد:

purchase.js

```
define(["credits","products"], function(credits,products) {
  console.log("Function : purchaseProduct");
  return {
    purchaseProduct: function() {
      var credit = credits.getCredits();
      if(credit > 0){
        products.reserveProduct();
        return true;
      }
      return false;
    }
  }
});
```

همان طور که مشاهده می‌کنید در فایل بالا از دستور define برای تعریف ماژول استفاده شده است و نام دو ماژول products و credits را به صورت پارامتر برای مشخص کردن وابستگی ماژول‌ها تعریف کردیم.

products.js

```
define(function(products) {
  return {
    reserveProduct: function() {
      console.log("Function : reserveProduct");
      return true;
    }
  }
});
```

credits.js

```
define(function() {
```

```
  console.log("Function : getCredits");
  return {
    getCredits: function() {
      var credits = "100";
      return credits;
    }
  }
});
```

به دلیل این که فایل‌های بالا وابستگی به ماژول‌های دیگر ندارند در نتیجه دستور define فقط شامل تعریف تابع است.
کافیست در فایل main.js، برای استفاده از فایل purchase.js از دستور require استفاده کنید.

```
require(['purchase'], function( purchase ) {
    var result = purchase.purchaseProduct();
});
```

مهم‌ترین مزیتی که این روش دارد این است که دیگر نیازی به نوشتن تگ‌های Script (آن هم به ترتیب درست) برای فراخوانی فایل‌های جاوااسکریپتی نخواهد بود؛ از طرفی دیگر وابستگی بین این فایل‌ها در هنگام تعریف ماژول مشخص خواهد شد. از آن به بعد وظیفه تامین فایل‌های مورد نیاز برای لود ماژول بر عهده RequireJs است.

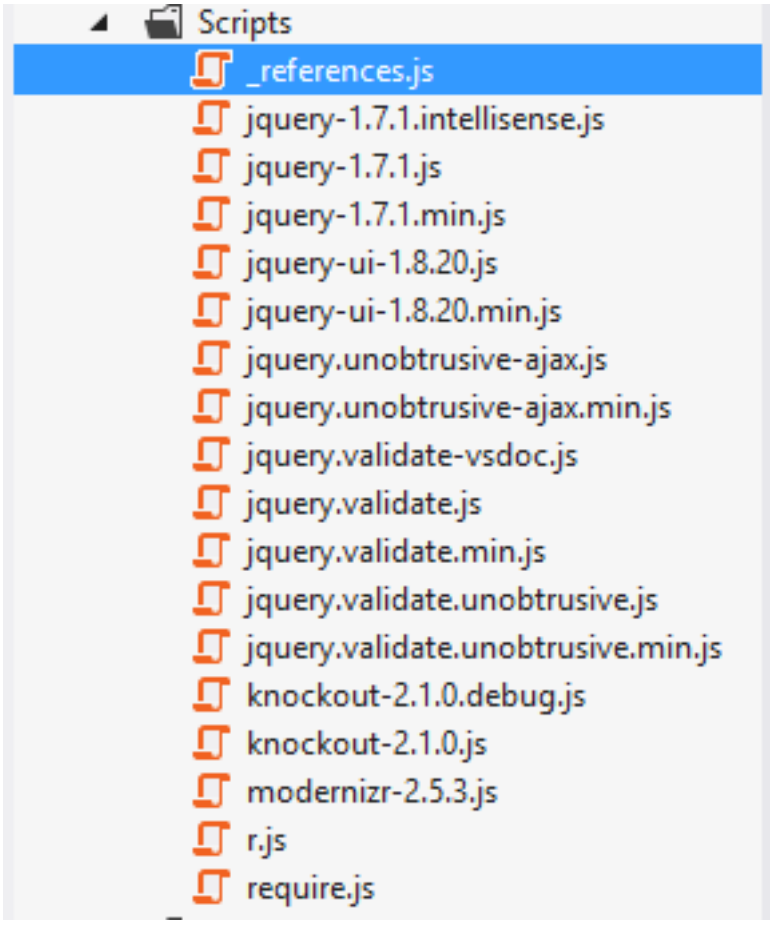
تفاوت بین دستور require و define
دستور define صرفا برای تعیین وابستگی ماژول استفاده می‌شود در حالی که دستور require برای فراخوانی و لود ماژول کاربرد دارد و به نوعی به عنوان نقطه شروع اجرای برنامه خواهد بود.
در پست بعدی به پیاده سازی مثال بالا به کمک RequireJs در قالب یک پروژه Asp.Net MVC خواهم پرداخت.

عنوان: **استفاده از RequireJs در پروژه های Asp.Net MVC**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۶/۳۰ ۸:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: requirejs, module

در پست قبلی با کلیات RequireJs آشنا شدید. در این به بررسی و پیاده سازی مثال قبل در قالب یک پروژه Asp.Net MVC می‌پردازم:
ابتدا یک پروژه Asp.Net MVC ایجاد کنید. در فولدر scripts تمام فایل‌های جاوااسکریپ پروژه قرار خواهند داشت. اگر قصد داشته باشیم که فایل‌های جاوااسکریپی سایر فریم ورک‌ها را استفاده نماییم (مثل backbone.js و ExtJs و...) برای طبقه بندی بهتر فایل ها، بهتر است که یک فولدر با نامی مشخص بسازیم و فایل‌های مورد نیاز را در آن قرار دهیم. البته اگر از nuget برای نصب این فریم ورک‌ها استفاده نمایید عموما این کار انجام خواهد شد.
حال با استفاده از Package Manager Console و اجرای دستور زیر، اقدام به نصب requireJs کنید

```
PM> Install-package requireJs
```

ساختار فولدر scripts به صورت زیر خواهد شد(دو فایل r.js و require.js به این فولدر اضافه می‌شود)



یک فولدر به نام MyFiles در فولدر Scripts بسازید و فایل‌های purchase.js و product.js و credits.js در پروژه قبل را در آن کپی نمایید. کد فایل‌های پروژه قبل به صورت زیر بوده است:
purchase.js

```
define(["credits","products"], function(credits,products) {
  console.log("Function : purchaseProduct");
  return {
```

```
    purchaseProduct: function() {
      var credit = credits.getCredits();
      if(credit > 0){
        products.reserveProduct();
        alert('purchase done');'
        return true;


 }
        alert('purchase cancel');
        return false;
    }
  }
});
```

در کد بالا از یک alert برای نمایش موفقیت یا عدم موفقیت عملیات استفاده کردم.

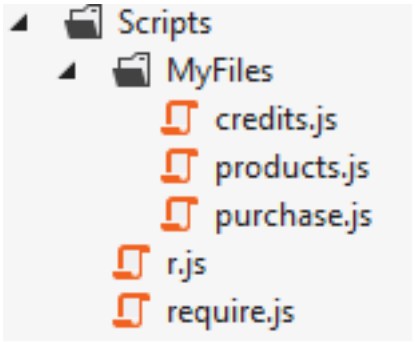
products.js

```
define(function(products) {
  return {
    reserveProduct: function() {
      console.log("Function : reserveProduct");
      return true;
    }
  }
});
```

credits.js

```
define(function() {
  console.log("Function : getCredits");
  return {
    getCredits: function() {
      var credits = "100";
      return credits;
    }
  }
});
```

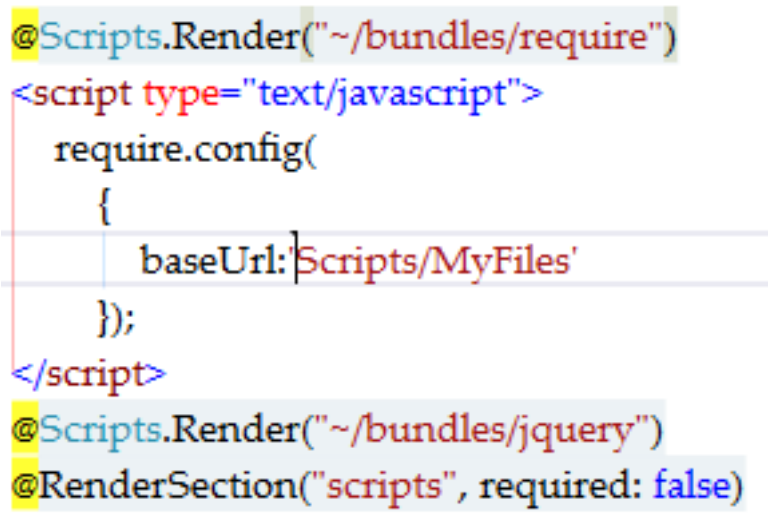
در نتیجه فایل‌های زیر به ساختار فولدر scripts اضافه شده است:



برای قدم بعدی، در متد RegisterBundles فایل bundleConfig پروژه دستور زیر را وارد نمایید:

```
bundles.Add( new ScriptBundle( "~/bundles/require" ).Include(
                      "~/Scripts/require.js" ) );
```

کاملا واضح است که نیاز به تغییر در فایل _Layout پروژه نیز داریم؛ در نتیجه تغییرات زیر را در فایل اعمال نمایید:

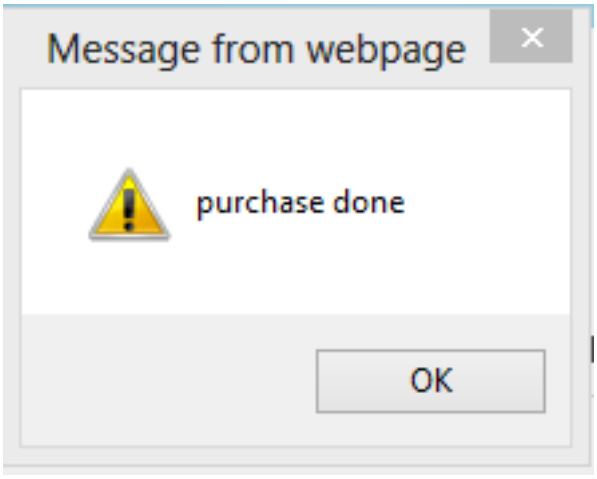


همان طور که مشاهده می‌کنید ابتدا با استفاده از دستور Scripts.Render فایل‌های include شده برای requireJs را در صفحه لود می‌کنید. سپس در تگ scripts که نوشته شده است با استفاده از دستور require.config مکان فایل‌های مورد نیاز را به فریم ورک Require معرفی میکنیم. این بدان معنی است که فریم ورک هر زمان که نیاز به لود یک وابستگی برای فایل‌های جاوااسکریپ داشته باشد، این مکان معرفی شده را جستجو خواهد کرد.

حال برای استفاده و لود ماژول purchase در انتهای فایل Index فولدر Home تغییرات زیر را اعمال نمایید:

```
@section scripts
{
    <script type="text/javascript">
    require(['purchase'], function (purchase)
    {
        purchase.purchaseProduct();
    });
</script>
}
```

در دستورات بالا با کمک دستور require(همان طور که در پست قبلی توضیح داده شد) ماژول purchase را لود می‌کنیم و بعد با فراخوانی تابع purchaseProduct به خروجی مورد نظر خواهیم رسید. در این جا من از دستور alert برای نمایش خروجی استفاده کردم! در نتیجه خروجی به صورت زیر خواهد بود:


Message from webpage
purchase done
OK