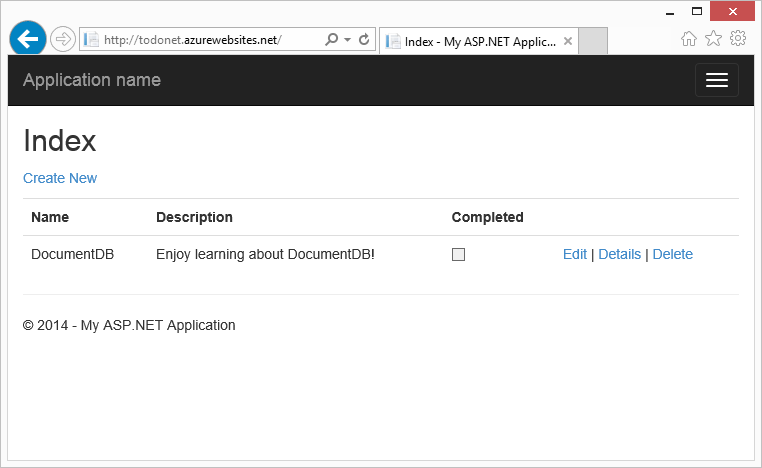
# ASP.NET MVC 教程：使用 DocumentDB 开发 Web 应用程序

[AZURE.SELECTOR] - [.NET](/documentation/articles/documentdb-dotnet-application) - [Node.js](/documentation/articles/documentdb-nodejs-application) - [Java](/documentation/articles/documentdb-java-application) - [Python](/documentation/articles/documentdb-python-application)

为了特别说明你可以如何有效地利用 Azure DocumentDB 来存储和查询 JSON 文档，本文提供了演示如何使用 Azure DocumentDB 构建待办事项应用的完整演练。任务将存储为 Azure DocumentDB 中的 JSON 文档。



屏幕截图：“ASP NET MVC 教程分步说明”教程创建的待办事项列表 MVC Web 应用程序

本演练演示如何使用 Azure 提供的 DocumentDB 服务来存储和访问 Azure 上托管的 ASP.NET MVC Web 应用程序的数据。

[AZURE.TIP] 本教程假定你先前有使用 ASP.NET MVC 和 Azure 网站的经验。如果你不熟悉 ASP.NET 或[必备工具](#_Toc395637760)，我们建议从 [GitHub](https://github.com/Azure-Samples/documentdb-net-todo-app) 下载完整的示例项目，并按照此示例中的说明操作。构建之后，你可以回顾本文以深入了解项目上下文中的代码。

## 本数据库教程的先决条件

在按照本文中的说明操作之前，你应确保已拥有下列项：

* 有效的 Azure 帐户。如果你没有帐户，只需花费几分钟就能创建一个免费试用帐户。有关详细信息，请参阅 [Azure 免费试用](/pricing/free-trial/)。
* [Visual Studio 2013](http://www.visualstudio.com/) Update 4 或更高版本。
* Azure SDK for .NET 2.5.1 版或更高版本，可通过 [Microsoft Web 平台安装程序](http://www.microsoft.com/web/downloads/platform.aspx)获取。

本文中的所有屏幕截图都是使用已应用 Update 4 的 Visual Studio 2013 以及 Azure SDK for .NET 2.5.1 版获取的。如果你的系统配备了不同的版本，那么，你的屏幕和选项可能不会完全相符，但只要你符合上述先决条件，本解决方案应该还是有效。

## 步骤 1：创建 DocumentDB 数据库帐户

让我们首先创建 DocumentDB 帐户。如果你已有帐户，则可以跳到[创建新的 ASP.NET MVC 应用程序](#_Toc395637762)。

[AZURE.INCLUDE [documentdb-create-dbaccount](../includes/documentdb-create-dbaccount.md)]

[AZURE.INCLUDE [documentdb-keys](../includes/documentdb-keys.md)]

现在，我们将演练如何从头开始创建新的 ASP.NET MVC 应用程序。

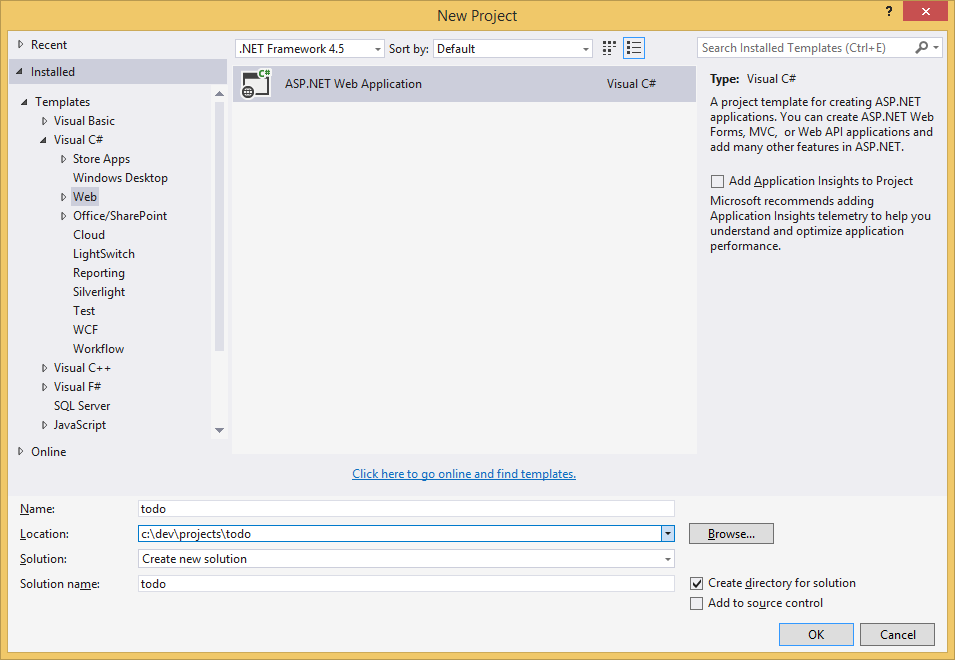
## 步骤 2：创建新的 ASP.NET MVC 应用程序

现在你已有帐户，我们可以开始创建新的 ASP.NET 项目。

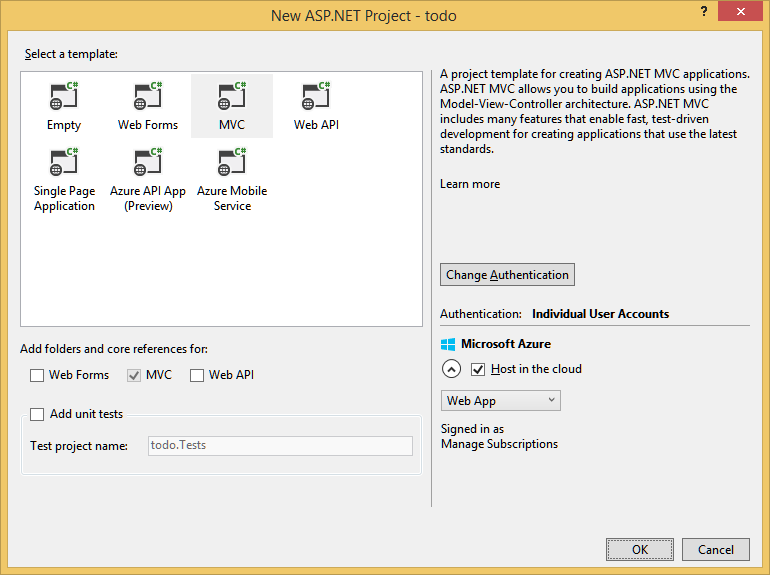
1. 在 Visual Studio 的“文件”菜单中，指向“新建”，然后单击“项目”。

* 将显示“新建项目”对话框。

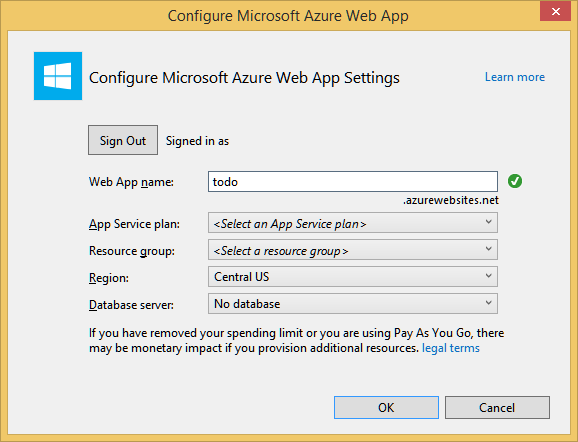
1. 在“项目类型”窗格中，依次展开“模板”、“Visual C#”、“Web”，然后选择“ASP.NET Web 应用程序”。

* 
* 屏幕截图：突出显示 ASP.NET Web 应用程序项目类型的“新建项目”对话框

1. 在“名称”框中，键入项目的名称。本教程使用名称“todo”。如果你选择使用其他名称，则每当本教程提及 todo 命名空间时，你必须调整所提供的代码示例，以便使用你为应用程序命名的名称。
2. 单击“浏览”以导航到你想在其中创建项目的文件夹，然后单击“确定”。

* 此时将显示“新建 ASP.NET 项目”对话框。
* 
* 屏幕截图：突出显示 MVC 应用程序模板并选中了“在云中托管”框的“新建 ASP.NET 项目”对话框

1. 在模板窗格中，选择“MVC”。
2. 如果你打算在 Azure 中托管应用程序，则选择右下角的“在云中托管”，让 Azure 托管应用程序。我们已选择要在云中托管，并运行 Azure 网站中托管的应用程序。选择此选项将会预先预配 Azure 网站，让你在需要部署最终的工作应用程序时更为容易。如果你希望在其他位置托管或者不想预先配置 Azure，只需清除“在云中托管”。
3. 单击“确定”，让 Visual Studio 围绕空白 ASP.NET MVC 模板基架的搭建执行操作。
4. 如果你选择在云中托管，则会出现至少一个附加屏幕，要求你登录 Azure 帐户并提供新网站的部分值。提供所有附加值，然后继续操作。

* 我在此处没有选择“数据库服务器”，因为我们并未使用 Azure SQL Database 服务器，稍后我们会在 Azure 门户中创建新的 Azure DocumentDB 帐户。
* 有关选择 **App Service 计划**和**资源组**的详细信息，请参阅 [Azure App Service 计划深入概述](../app-service/azure-web-sites-web-hosting-plans-in-depth-overview.md)。
* 
* 屏幕截图：“配置 Microsoft Azure 网站”对话框

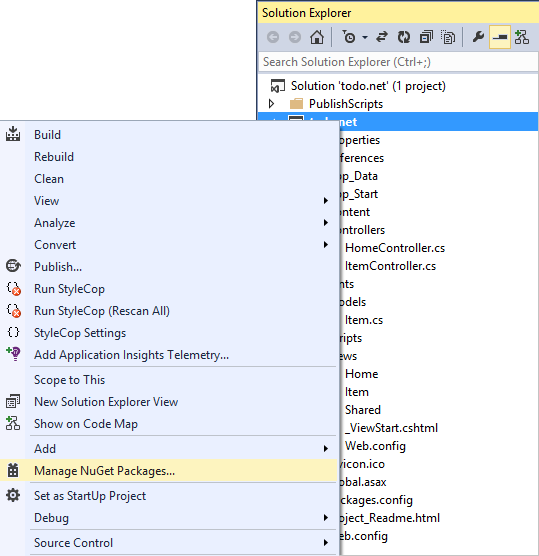
1. Visual Studio 创建好样板 MVC 应用程序之后，你便拥有可以在本地运行的空白 ASP.NET 应用程序。

* 我们会跳过在本地运行项目，因为我确定我们都已看过 ASP.NET“Hello World”应用程序。让我们直接跳到将 DocumentDB 添加到此项目并构建应用程序的步骤。

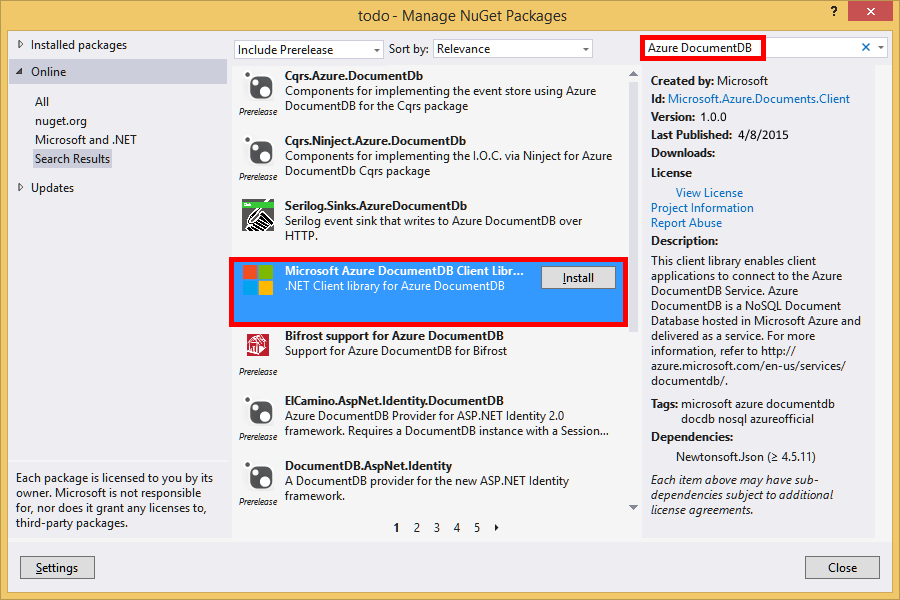
## 步骤 3：将 DocumentDB 添加到 MVC Web 应用程序项目

我们已经完成了此解决方案的大部分 ASP.NET MVC 琐事，现在可以开始本教程的真正目的，也就是将 Azure DocumentDB 添加到 MVC Web 应用程序。

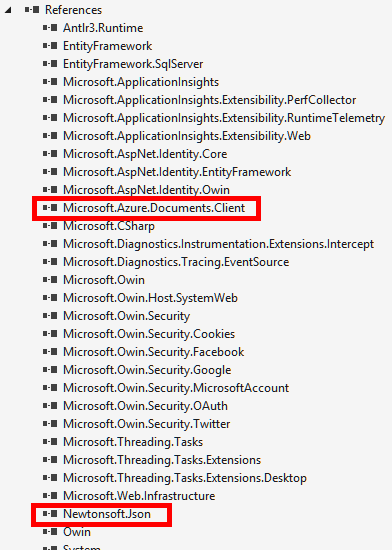
1. DocumentDB .NET SDK 将打包并以 NuGet 程序包的形式分发。若要在 Visual Studio 中获取 NuGet 程序包，请使用 Visual Studio 中的 NuGet 程序包管理器，方法是右键单击“解决方案资源管理器”中的项目，然后单击“管理 NuGet 程序包”。

* 
* 屏幕截图：解决方案资源管理器中 Web 应用程序项目的右键单击选项，其中突出显示了“管理 NuGet 程序包”。
* 此时将显示“管理 NuGet 包”对话框。

1. 在“联机搜索”框中，键入 **Azure DocumentDB**。

* 从结果中安装“Microsoft Azure DocumentDB 客户端库”程序包。这将下载并安装 DocumentDB 程序包，以及所有依赖项（例如 Newtonsoft.Json）。
* 
* 屏幕截图：突出显示 Microsoft Azure DocumentDB 客户端库的“管理 NuGet 程序包”窗口
* 或者，你也可以使用程序包管理器控制台来安装程序包。为此，请在“工具”菜单中，单击“Nuget 程序包管理器”，然后单击“程序包管理器控制台”。在提示符处键入以下命令。
* Install-Package Microsoft.Azure.DocumentDB

1. 安装程序包之后，你的 Visual Studio 解决方案应该类似于下列添加了两个新引用（Microsoft.Azure.Documents.Client 和 Newtonsoft.Json）的解决方案。

* 
* 屏幕截图：解决方案资源管理器中添加到 JSON 数据项目的两个引用

## 步骤 4：设置 ASP.NET MVC 应用程序

现在我们可以开始向此 MVC 应用程序添加模型、视图和控制器：

* [添加模型](#_Toc395637764)。
* [添加控制器](#_Toc395637765)。
* [添加视图](#_Toc395637766)。

### 添加 JSON 数据模型

首先，让我们在 MVC 中创建 **M**（模型）。

1. 在“解决方案资源管理器”中，右键单击“Models”文件夹，单击“添加”，然后单击“类”。

* 此时将显示“添加新项”对话框。

1. 将新类命名为 **Item.cs**，然后单击“添加”。
2. 在这个新的 **Item.cs** 文件中，将下列代码添加到最后一个 using 语句后面。

* using Newtonsoft.Json;

1. 现在将此代码

* public class Item  
  {  
  }
* 替换为以下代码。
* public class Item  
  {  
   [JsonProperty(PropertyName = "id")]  
   public string Id { get; set; }  
    
   [JsonProperty(PropertyName = "name")]  
   public string Name { get; set; }  
    
   [JsonProperty(PropertyName = "description")]  
   public string Description { get; set; }  
    
   [JsonProperty(PropertyName = "isComplete")]  
   public bool Completed { get; set; }  
  }
* DocumentDB 中的所有数据都会通过线路传递，并存储为 JSON。若要通过 JSON.NET 控制对象的序列化/反序列化方式，可以使用刚才创建的**项**类中所示的 **JsonProperty** 属性。你无**需**这样做，但我想确保所有属性都按照 JSON camelCase 命名约定命名。
* 使用 JSON 时，你不但可以控制属性名称的格式，还可以像我命名 **Description** 属性一样重命名你的 .NET 属性。

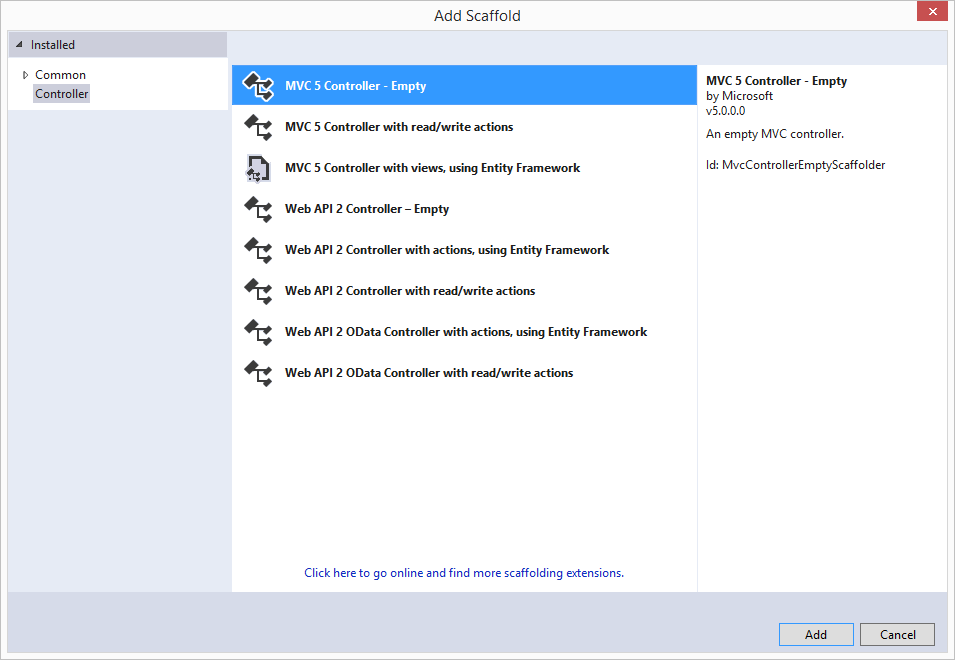
### 添加控制器

我们已经创建了 **M**，现在让我们创建 MVC 中的 **C**（控制器类）。

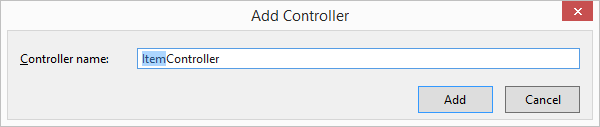
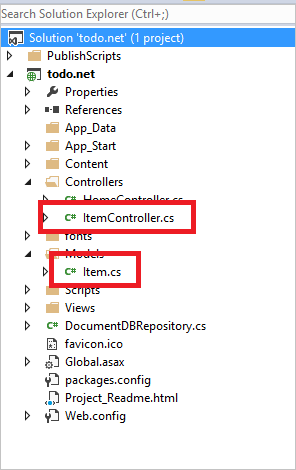
1. 在“解决方案资源管理器”中，右键单击“Controllers”文件夹，单击“添加”，然后单击“控制器”。

* 此时将显示“添加基架”对话框。

1. 选择“MVC 5 控制器 - 空”，然后单击“添加”。

* 
* 屏幕截图：突出显示“MVC 5 控制器 - 空”选项的“添加基架”对话框

1. 将新控制器命名为 **ItemController**。

* 
* 屏幕截图：“添加控制器”对话框
* 创建文件之后，你的 Visual Studio 解决方案应该类似于下列在“解决方案资源管理器”中包含有新 ItemController.cs 文件的解决方案。系统还将显示先前创建的新 Item.cs 文件。
* 
* 屏幕截图：Visual Studio 解决方案 — 突出显示新 ItemController.cs 文件和 Item.cs 文件的解决方案资源管理器
* 你可以关闭 ItemController.cs，我们稍后会回头使用此文件。

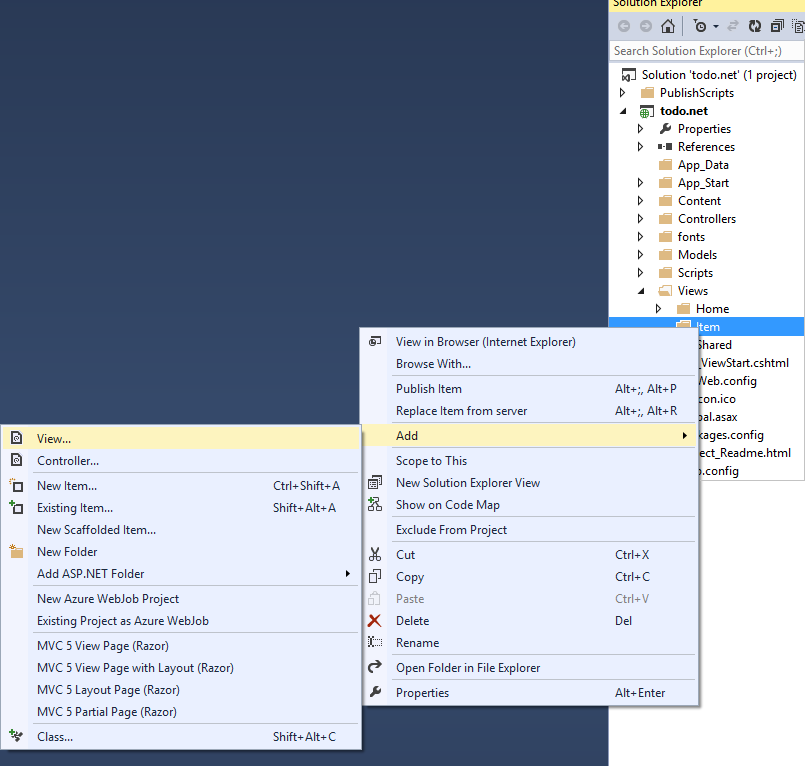
### 添加视图

现在，我们可以开始创建 MVC 中的 **V**（视图）：

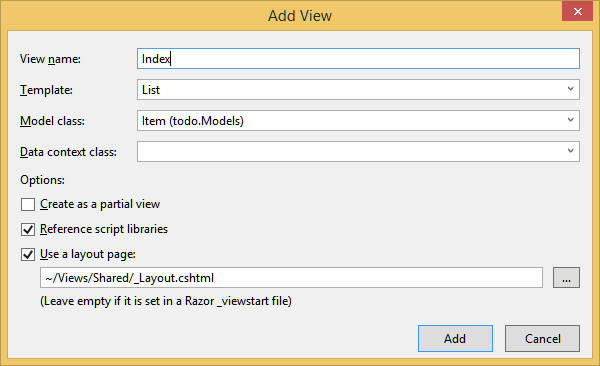
* [添加“项索引”视图](#AddItemIndexView)。
* [添加“新建项”视图](#AddNewIndexView)。
* [添加“编辑项”视图](#_Toc395888515)。

#### 添加“项索引”视图

1. 在“解决方案资源管理器”中，展开“Views”文件夹，右键单击先前你在添加 **ItemController** 时，Visual Studio 为你创建的空白“Item”文件夹，单击“添加”，然后单击“视图”。

* 
* 屏幕截图：显示 Visual Studio 使用突出显示的“添加视图”命令创建的 Item 文件夹的解决方案资源管理器

1. 在“添加视图”对话框中，执行以下操作：
   * 在“视图名称”框中，键入“索引”。
   * 在“模板”框中，选择“列表”。
   * 在“模型类”框中，选择“项(todo.Models)”。
   * 将“数据上下文类”框留空。
   * 在“布局页”框中，键入 **~/Views/Shared/\_Layout.cshtml**。

* 
* 屏幕截图：显示“添加视图”对话框

1. 设置完所有这些值之后，单击“添加”，让 Visual Studio 创建新的模板视图。完成之后，它会打开刚创建的 cshtml 文件。我们可以在 Visual Studio 中关闭该文件，我们稍后会回头使用此文件。

#### 添加“新建项”视图

与创建“项索引”视图的方式类似，我们现在可以开始创建新的视图，以供创建新**项**使用。

1. 在“解决方案资源管理器”中，再次右键单击“Item”文件夹，单击“添加”，然后单击“视图”。
2. 在“添加视图”对话框中，执行以下操作：
   * 在“视图名称”框中，键入“创建”。
   * 在“模板”框中，选择“创建”。
   * 在“模型类”框中，选择“项(todo.Models)”。
   * 将“数据上下文类”框留空。
   * 在“布局页”框中，键入 **~/Views/Shared/\_Layout.cshtml**。
   * 单击**“添加”**。

#### 添加“编辑项”视图

最后，采用与之前相同的方式添加最后一个视图，以供编辑**项**使用。

1. 在“解决方案资源管理器”中，再次右键单击“Item”文件夹，单击“添加”，然后单击“视图”。
2. 在“添加视图”对话框中，执行以下操作：
   * 在“视图名称”框中，键入“编辑”。
   * 在“模板”框中，选择“编辑”。
   * 在“模型类”框中，选择“项(todo.Models)”。
   * 将“数据上下文类”框留空。
   * 在“布局页”框中，键入 **~/Views/Shared/\_Layout.cshtml**。
   * 单击**“添加”**。

完成此操作之后，关闭 Visual Studio 中的所有 cshtml 文档，我们稍后会回头使用这些视图。

## 步骤 5：连接 DocumentDB

我们已经创建了标准的 MVC 项目，现在我们可以开始添加 DocumentDB 的代码。

在本节中，我们将添加代码来处理下列操作：

* [列出未完成的项](#_Toc395637770)。
* [添加项](#_Toc395637771)。
* [编辑项](#_Toc395637772)。

### 列出 MVC Web 应用程序中未完成的项

首先要执行的操作是添加类，其中包含要连接并使用 DocumentDB 的所有逻辑。在本教程中，我们会将所有逻辑封装到名为 DocumentDBRepository 的存储库类中。

1. 在“解决方案资源管理器”中，右键单击该项目，单击“添加”，然后单击“类”。将新类命名为 **DocumentDBRepository**，然后单击“添加”。
2. 在刚刚创建的 **DocumentDBRepository** 类中，在命名空间声明上方添加下列 using 语句

* using Microsoft.Azure.Documents;   
  using Microsoft.Azure.Documents.Client;   
  using Microsoft.Azure.Documents.Linq;   
  using System.Configuration;  
  using System.Linq.Expressions;  
  using System.Threading.Tasks;
* 现在将此代码
* public class DocumentDBRepository  
  {  
  }
* 替换为以下代码。
* public static class DocumentDBRepository<T> where T : class  
  {  
   private static readonly string DatabaseId = ConfigurationManager.AppSettings["database"];  
   private static readonly string CollectionId = ConfigurationManager.AppSettings["collection"];  
   private static DocumentClient client;  
    
   public static void Initialize()  
   {  
   client = new DocumentClient(new Uri(ConfigurationManager.AppSettings["endpoint"]), ConfigurationManager.AppSettings["authKey"]);  
   CreateDatabaseIfNotExistsAsync().Wait();  
   CreateCollectionIfNotExistsAsync().Wait();  
   }  
    
   private static async Task CreateDatabaseIfNotExistsAsync()  
   {  
   try  
   {  
   await client.ReadDatabaseAsync(UriFactory.CreateDatabaseUri(DatabaseId));  
   }  
   catch (DocumentClientException e)  
   {  
   if (e.StatusCode == System.Net.HttpStatusCode.NotFound)  
   {  
   await client.CreateDatabaseAsync(new Database { Id = DatabaseId });  
   }  
   else  
   {  
   throw;  
   }  
   }  
   }  
    
   private static async Task CreateCollectionIfNotExistsAsync()  
   {  
   try  
   {  
   await client.ReadDocumentCollectionAsync(UriFactory.CreateDocumentCollectionUri(DatabaseId, CollectionId));  
   }  
   catch (DocumentClientException e)  
   {  
   if (e.StatusCode == System.Net.HttpStatusCode.NotFound)  
   {  
   await client.CreateDocumentCollectionAsync(  
   UriFactory.CreateDatabaseUri(DatabaseId),  
   new DocumentCollection { Id = CollectionId },  
   new RequestOptions { OfferThroughput = 1000 });  
   }  
   else  
   {  
   throw;  
   }  
   }  
   }  
  }
* [AZURE.TIP] 创建新的 DocumentCollection 时，你可以提供 OfferType 的可选 RequestOptions 参数，此参数可让你指定新集合的性能级别。如果未传递此参数，系统将使用默认的产品/服务类型。有关 DocumentDB 产品/服务类型的详细信息，请参阅 [DocumentDB 性能级别](/documentation/articles/documentdb-performance-levels.md)

1. 我们打算从配置中读取部分值，因此请打开应用程序的 **Web.config** 文件，并在 <AppSettings> 节下面添加下列几行。

* <add key="endpoint" value="enter the URI from the Keys blade of the Azure Portal"/>  
  <add key="authKey" value="enter the PRIMARY KEY, or the SECONDARY KEY, from the Keys blade of the Azure Portal"/>  
  <add key="database" value="ToDoList"/>  
  <add key="collection" value="Items"/>

1. 现在，使用 Azure 门户的“密钥”边栏选项卡来更新终结点和 authKey 的值。使用“密钥”边栏选项卡中的“URI”作为终结点设置的值，使用“密钥”边栏选项卡中的“主密钥”或“辅助密钥”作为 authKey 设置的值。

* 我们已经连接了 DocumentDB 存储库，现在让我们添加应用程序逻辑。

1. 我们想要用待办事项列表应用程序做的第一件事就是显示未完成的项。在 **DocumentDBRepository** 类中的任意位置复制并粘贴下列代码段。

* public static async Task<IEnumerable<T>> GetItemsAsync(Expression<Func<T, bool>> predicate)  
  {  
   IDocumentQuery<T> query = client.CreateDocumentQuery<T>(  
   UriFactory.CreateDocumentCollectionUri(DatabaseId, CollectionId))  
   .Where(predicate)  
   .AsDocumentQuery();  
    
   List<T> results = new List<T>();  
   while (query.HasMoreResults)  
   {  
   results.AddRange(await query.ExecuteNextAsync<T>());  
   }  
    
   return results;  
  }

1. 打开我们先前添加的 **ItemController**，并在命名空间声明上方添加下列 using 语句。

* using System.Net;  
  using System.Threading.Tasks;  
  using todo.Models;
* 如果你的项目名称不是“todo”，则必须使用“todo. Models”进行更新，才能反映你的项目名称。
* 现在将此代码
* //GET: Item  
  public ActionResult Index()  
  {  
   return View();  
  }
* 替换为以下代码。
* [ActionName("Index")]  
  public async Task<ActionResult> IndexAsync()  
  {  
   var items = await DocumentDBRepository<Item>.GetItemsAsync(d => !d.Completed);  
   return View(items);  
  }

此时，应该可以构建解决方案，而不会发生任何错误。

如果现在运行应用程序，你可以转到 **HomeController** 以及该控制器的“索引”视图。这是我们一开始就选定的 MVC 模板项目默认行为，但是我们不想要这样的行为！ 让我们更改此 MVC 应用程序上的路由以改变此行为。

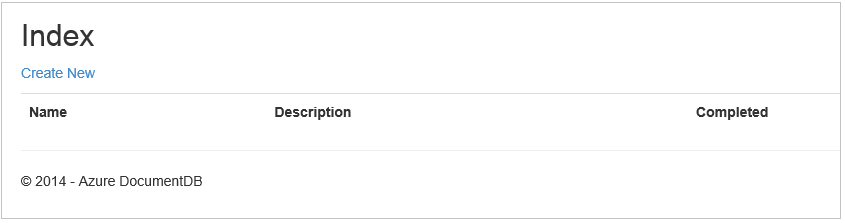
打开 **App\_Start\RouteConfig.cs**，找到以“defaults:”开头的行，然后将它更改为如下行。

defaults: new { controller = "Item", action = "Index", id = UrlParameter.Optional }

如果你未在 URL 中指定控制路由行为的值，这会让 ASP.NET MVC 知道改用 **Item**（而不是 **Home**）作为控制器，并使用**索引**作为视图。

如果你运行应用程序，它现在会调用到你的 **ItemController**，进而调用到存储库类，并使用 GetItems 方法将所有未完成的项返回到“视图\项\索引”视图。

如果构建并立即运行此项目，你现在应该会看到如下内容。



屏幕截图：本数据库教程创建的待办事项列表 Web 应用程序

### 添加项

我们可以开始将一些项放入数据库中，所以除了空白网格以外，我们还可以看到其他内容。

让我们将一些代码添加到 DocumentDBRepository 和 ItemController，以在 DocumentDB 中保留记录。

1. 将下列方法添加到 **DocumentDBRepository** 类。

* public static async Task<Document> CreateItemAsync(T item)  
  {  
   return await client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(DatabaseId, CollectionId), item);  
  }
* 此方法只接受传递给它的对象，并将对象保留在 DocumentDB 中。

1. 打开 ItemController.cs 文件，并在类中添加下列代码段。这是 ASP.NET MVC 得知如何执行**创建**操作的方式。在此情况下，只需呈现先前创建的关联 Create.cshtml 视图。

* [ActionName("Create")]  
  public async Task<ActionResult> CreateAsync()  
  {  
   return View();  
  }
* 现在此控制器需要更多代码，以接受“创建”视图所提交的数据。

1. 将下一个代码块添加到 ItemController.cs 类，以告诉 ASP.NET MVC 如何使用窗体 POST 来执行此控制器的操作。

* [HttpPost]  
  [ActionName("Create")]  
  [ValidateAntiForgeryToken]  
  public async Task<ActionResult> CreateAsync([Bind(Include = "Id,Name,Description,Completed")] Item item)  
  {  
   if (ModelState.IsValid)  
   {  
   await DocumentDBRepository<Item>.CreateItemAsync(item);  
   return RedirectToAction("Index");  
   }  
    
   return View(item);  
  }
* 此代码会调用到 DocumentDBRepository，并使用 CreateItemAsync 方法将新的待办事项保存到数据库。
* **安全说明**：此处所使用的 **ValidateAntiForgeryToken** 属性可帮助此应用程序防止跨网站请求伪造攻击。这不仅仅是添加此属性，你的视图也必须使用此防伪令牌。有关此主题的详细信息以及如何正确实施此操作的示例，请参阅[防止跨网站请求伪造](http://go.microsoft.com/fwlink/?LinkID=517254)。[GitHub](https://github.com/Azure-Samples/documentdb-net-todo-app) 上提供的源代码已有完整实现。
* **安全说明**：我们也会在方法参数上使用 **Bind** 属性，以帮助防范 over-posting 攻击。有关更多详细信息，请参阅 [ASP.NET MVC 中的基本 CRUD 操作](http://go.microsoft.com/fwlink/?LinkId=317598)。

将新项添加到数据库所需的代码至此结束。

### 编辑项

我们最后还要做一件事，那就是添加在数据库中编辑**项**并将它们标记为已完成的功能。编辑视图已添加到项目中，因此，我们只需重新将某些代码添加到控制器和 **DocumentDBRepository** 类。

1. 将下列代码添加到 **DocumentDBRepository** 类。

* public static async Task<Document> UpdateItemAsync(string id, T item)  
  {  
   return await client.ReplaceDocumentAsync(UriFactory.CreateDocumentUri(DatabaseId, CollectionId, id), item);  
  }  
    
  public static async Task<T> GetItemAsync(string id)  
  {  
   try  
   {  
   Document document = await client.ReadDocumentAsync(UriFactory.CreateDocumentUri(DatabaseId, CollectionId, id));  
   return (T)(dynamic)document;  
   }  
   catch (DocumentClientException e)  
   {  
   if (e.StatusCode == HttpStatusCode.NotFound)  
   {  
   return null;  
   }  
   else  
   {  
   throw;  
   }  
   }  
  }
* 这些方法中的第一个方法 (**GetItem**) 会从 DocumentDB 中提取某个项，此项会被传递回 **ItemController**，接着传到“编辑”视图。
* 刚刚添加的第二个方法使用从 **ItemController** 传入的**文档**版本取代 DocumentDB 中的**文档**。

1. 将下列代码添加到 **ItemController** 类。

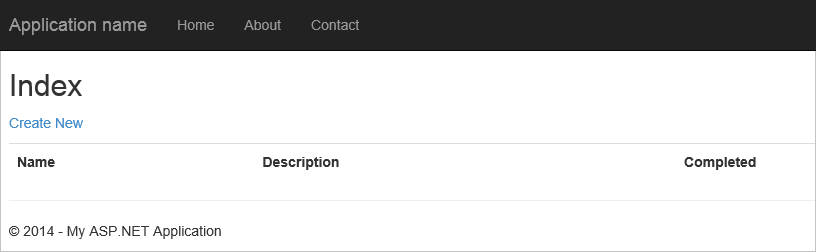
* [HttpPost]  
  [ActionName("Edit")]  
  [ValidateAntiForgeryToken]  
  public async Task<ActionResult> EditAsync([Bind(Include = "Id,Name,Description,Completed")] Item item)  
  {  
   if (ModelState.IsValid)  
   {  
   await DocumentDBRepository<Item>.UpdateItemAsync(item.Id, item);  
   return RedirectToAction("Index");  
   }  
    
   return View(item);  
  }  
    
  [ActionName("Edit")]  
  public async Task<ActionResult> EditAsync(string id)  
  {  
   if (id == null)  
   {  
   return new HttpStatusCodeResult(HttpStatusCode.BadRequest);  
   }  
    
   Item item = await DocumentDBRepository<Item>.GetItemAsync(id);  
   if (item == null)  
   {  
   return HttpNotFound();  
   }  
    
   return View(item);  
  }
* 第一个方法会处理当用户单击“索引”视图中的“编辑”链接时所发生的 Http GET。此方法会从 DocumentDB 中提取[**文档**](http://msdn.microsoft.com/library/azure/microsoft.azure.documents.document.aspx)，并将它传递给“编辑”视图。
* “编辑”视图会接着对 **IndexController** 执行 Http POST 操作。
* 添加的第二个方法会处理此操作：将更新对象传递到 DocumentDB 以便保留在数据库中。

这样便大功告成了，这些就是我们必须运行应用程序的所有操作：列出未完成的**项**，添加新**项**，最后是编辑**项**。

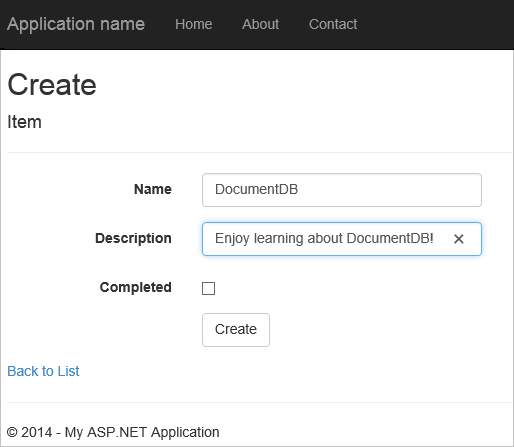
## 步骤 6：在本地运行应用程序

若要在本地计算机上测试应用程序，请执行以下操作：

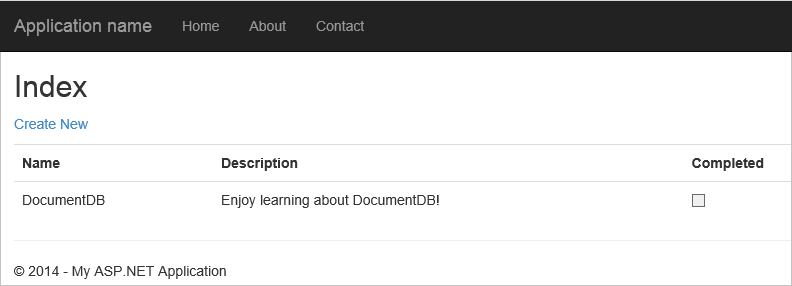
1. 在 Visual Studio 中按 F5，即可在调试模式下构建应用程序。这样应该可以构建应用程序，并启动包含先前看到的空白网格页面的浏览器：

* 
* 屏幕截图：本数据库教程创建的待办事项列表 Web 应用程序
* 如果此时发生错误，你可以将代码与 [GitHub](https://github.com/Azure-Samples/documentdb-net-todo-app) 上的示例项目进行比较

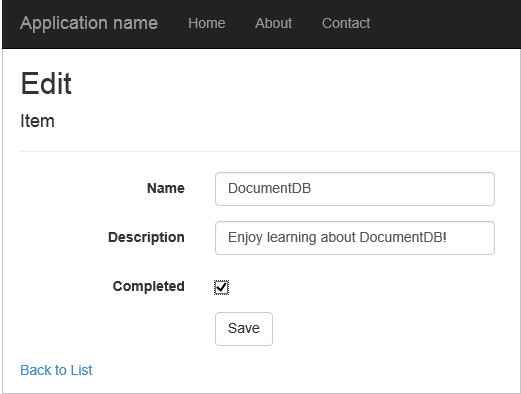
1. 单击“新建”链接，并在“名称”和“描述”字段中添加值。将“已完成”复选框保持为未选中状态，否则，新**项**将以已完成状态添加，不会出现在初始列表中。

* 
* 屏幕截图：“创建”视图

1. 单击“创建”，你将被重定向回“索引”视图，并且你的**项**会出现在列表中。

* 
* 屏幕截图：“索引”视图
* 向待办事项列表任意添加更多**项**。

1. 单击列表上某个**项**旁边的“编辑”，你将转至“编辑”视图，你可以在此更新对象的任何属性（包括“已完成”标志）。如果你标记“已完成”标志并单击“保存”，该**项**将从未完成任务列表中删除。

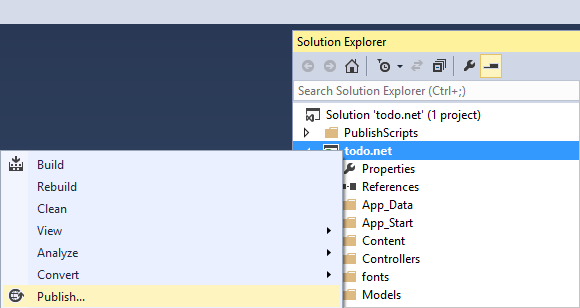
* 
* 屏幕截图：选中了“已完成”框的“索引”视图

1. 完成应用测试后，按 Ctrl+F5 停止调试应用。你可以开始部署了！

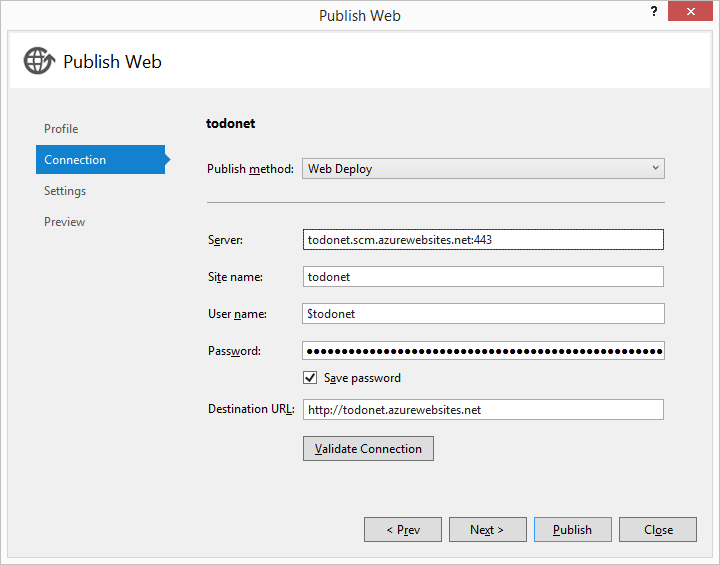
## 步骤 7：将应用程序部署到 Azure 网站

你已经拥有可在 DocumentDB 正常工作的完整应用程序，我们现在要将此 Web 应用部署到 Azure 网站。如果你在创建空白 ASP.NET MVC 项目时选择了“在云中托管”，则 Visual Studio 可让这项操作变得十分简单，并为你完成大部分的任务。

1. 若要发布此应用程序，你只需要右键单击“解决方案资源管理器”中的项目，然后单击“发布”。

* 
* 屏幕截图：解决方案资源管理器中的“发布”选项

1. 系统应该已根据你的凭据配置好所有项目；实际上，系统已在 Azure 中创建位于所示**目标 URL** 的网站，你只需单击“发布”即可。

* 
* 屏幕截图：Visual Studio 中的“发布 Web”对话框 — ASP NET MVC 教程分步说明

在几秒钟内，Visual Studio 将完成 Web 应用程序发布并启动浏览器，你可从中查看在 Azure 中运行的简单作品！

## 后续步骤

祝贺你！ 你刚使用 Azure DocumentDB 构建了第一个 ASP.NET MVC 应用程序并将其发布到了 Azure 网站。你可以从 [GitHub](https://github.com/Azure-Samples/documentdb-net-todo-app) 下载或克隆完整应用程序（包括本教程未涵盖的详细信息和删除功能）的源代码。因此，如果你想将代码添加到应用中，请捕捉代码，再将它添加到此应用中。

若要向应用程序添加其他功能，请查看 [DocumentDB .NET 库](https://msdn.microsoft.com/library/azure/dn948556.aspx)中提供的 API，并欢迎你贡献到 [GitHub](https://github.com/Azure-Samples/documentdb-net-todo-app) 上的 DocumentDB .NET 库。