# 使用 Azure DocumentDB 中的地理空间数据

本文将介绍 [Azure DocumentDB](/services/documentdb/) 中的地理空间功能。在阅读本文之后，你将能够回答以下问题：

* 我如何在 Azure DocumentDB 中存储空间数据？
* 我如何使用 SQL 和 LINQ 查询 Azure DocumentDB 中的地理空间数据？
* 我如何在 DocumentDB 中启用或禁用空间索引？

请参阅此 [Github 项目](https://github.com/Azure/azure-documentdb-dotnet/blob/master/samples/code-samples/Geospatial/Program.cs)中的代码示例。

## 空间数据简介

空间数据用于描述空间中对象的位置和形状。在大部分应用程序中，这些会对应于地球上的对象，也就是地理空间数据。空间数据可以用来表示人、名胜古迹、城市边界或湖泊所处的位置。常见用例通常涉及邻近查询，例如“寻找我目前位置附近的所有咖啡厅”。

### GeoJSON

DocumentDB 支持对使用 [GeoJSON 规范](http://geojson.org/geojson-spec.html)表示的地理空间点数据进行索引编制和查询。GeoJSON 数据结构永远是有效的 JSON 对象，因此可以通过 DocumentDB 来存储和查询，无需任何专用的工具或库。DocumentDB SDK 提供帮助程序类和方法，让空间数据更易使用。

### 点、LineString 和多边形

**点**代表空间中的单一位置。在地理空间数据中，某个点所代表的确切位置可能是杂货店、电话亭、汽车或城市的街道地址。点使用其坐标对或经纬度，以 GeoJSON 格式（和 DocumentDB）表示。以下是点的 JSON 示例。

**DocumentDB 中的点**

{  
 "type":"Point",  
 "coordinates":[ 31.9, -4.8 ]  
}

[AZURE.NOTE] GeoJSON 规范先指定经度，再指定纬度。与其他地图应用程序一样，经度和纬度为角度，并以度为单位表示。经度值从本初子午线测量，并介于 -180 度和 180.0 度之间；纬度值从赤道测量，并介于 -90.0 度和 90.0 度之间。

DocumentDB 会将坐标解释为按照 WGS-84 参考系统表示。有关坐标参考系统的更多详细信息，请参阅下文。

如下面包含位置数据的用户配置文件示例所示，此数据可以嵌入 DocumentDB 文档中：

**存储在 DocumentDB 中包含位置的用户配置文件**

{  
 "id":"documentdb-profile",  
 "screen\_name":"@DocumentDB",  
 "city":"Redmond",  
 "topics":[ "NoSQL", "Javascript" ],  
 "location":{  
 "type":"Point",  
 "coordinates":[ 31.9, -4.8 ]  
 }  
}

除了点之外，GeoJSON 也支持 LineString 和多边形。**LineString** 表示空间中一连串的点（两个或更多个）以及连接这些点的线段。在地理空间数据中，LineString 通常用来表示高速公路或河流。**多边形**是由相连接的点组成的边界，并形成闭合的 LineString。多边形通常用来表示自然构成物（例如湖泊），或表示政治管辖权（例如省/市/自治区）。以下是 DocumentDB 中多边形的示例。

**DocumentDB 中的多边形**

{  
 "type":"Polygon",  
 "coordinates":[  
 [ 31.8, -5 ],  
 [ 31.8, -4.7 ],  
 [ 32, -4.7 ],  
 [ 32, -5 ],  
 [ 31.8, -5 ]  
 ]  
}

[AZURE.NOTE] GeoJSON 规范需要此数据才能形成有效的多边形；若要创建一个闭合的形状，最后一个坐标对应该与第一个坐标对相同。

多边形内的点必须以逆时针顺序指定。以顺时针顺序指定的多边形表示其中的区域倒转。

除了点、LineString 和多边形之外，GeoJSON 也会针对如何对多个地理空间位置分组以及如何将任意属性与地理位置关联成为**特征**指定表示形式。由于这些对象都是有效的 JSON，因此均可在 DocumentDB 中存储及处理。不过，DocumentDB 仅支持自动编制点的索引。

### 坐标参考系统

由于地球的形状不规则，地理空间数据的坐标可以用许多坐标参考系统 (CRS) 来表示，而这些系统各有自己的参考框架和测量单位。例如，“英国国家网格”对英国而言是非常精确的参考系统，但对其他地区则不是。

现今最常使用的 CRS 是世界测地系统 [WGS-84](http://earth-info.nga.mil/GandG/wgs84/)。GPS 设备和许多地图服务（包括谷歌地图和必应地图 API）均使用 WGS-84。DocumentDB 仅支持对使用 WGS-84 CRS 的地理空间数据进行索引编制和查询。

## 创建包含空间数据的文档

创建包含 GeoJSON 值的文档时，值会根据集合的索引策略，自动以空间索引进行索引编制。如果以动态类型化语言（如 Python 或 Node.js）使用 DocumentDB SDK，则必须创建有效的 GeoJSON。

**在 Node.js 中创建包含地理空间数据的文档**

var userProfileDocument = {  
 "name":"documentdb",  
 "location":{  
 "type":"Point",  
 "coordinates":[ -122.12, 47.66 ]  
 }  
};  
  
client.createDocument(`dbs/${databaseName}/colls/${collectionName}`, userProfileDocument, (err, created) => {  
 // additional code within the callback  
});

如果使用 .NET（或 Java）SDK，则可以在 Microsoft.Azure.Documents.Spatial 命名空间中使用新的点和多边形类，将位置信息嵌入应用程序对象中。这些类有助于简化将空间数据序列化和反序列化为 GeoJSON 的过程。

**在 .NET 中创建包含地理空间数据的文档**

using Microsoft.Azure.Documents.Spatial;  
  
public class UserProfile  
{  
 [JsonProperty("name")]  
 public string Name { get; set; }  
  
 [JsonProperty("location")]  
 public Point Location { get; set; }  
   
 // More properties  
}  
  
await client.CreateDocumentAsync(  
 UriFactory.CreateDocumentCollectionUri("db", "profiles"),   
 new UserProfile   
 {   
 Name = "documentdb",   
 Location = new Point (-122.12, 47.66)   
 });

如果你没有经纬度信息，但有物理地址或位置名称，如城市或国家/地区，则可以使用必应地图 REST 服务等地理编码服务来查找实际的坐标。在[此处](https://msdn.microsoft.com/library/ff701713.aspx)详细了解必应地图地理编码。

## 查询空间类型

我们已经探讨过如何插入地理空间数据，现在就来看看如何通过 SQL 和 LINQ 使用 DocumentDB 查询此数据。

### 空间 SQL 内置函数

DocumentDB 支持以下用于查询地理空间的开放地理空间信息联盟 (OGC) 内置函数。有关 SQL 语言中的整套内置函数的更多详细信息，请参阅[查询 DocumentDB](/documentation/articles/documentdb-sql-query)。

使用情况

说明

ST\_DISTANCE (point\_expr、point\_expr)

返回两个 GeoJSON 点表达式之间的距离。

ST\_WITHIN (point\_expr、polygon\_expr)

返回布尔表达式，该表达式指示第一个参数中指定的 GeoJSON 点是否位于第二个参数中的 GeoJSON 多边形内。

ST\_ISVALID

返回一个布尔值，该值指示指定的 GeoJSON 点或多边形表达式是否有效。

ST\_ISVALIDDETAILED

如果指定的 GeoJSON 点或多边形表达式有效，则返回包含布尔值的 JSON 值；如果无效，则额外加上作为字符串值的原因。

空间函数可用于对空间数据执行邻近查询。例如，以下查询使用 ST\_DISTANCE 内置函数返回所有家族文档，且这些文档在指定位置的 30 公里内。

**查询**

SELECT f.id   
FROM Families f   
WHERE ST\_DISTANCE(f.location, {'type': 'Point', 'coordinates':[31.9, -4.8]}) < 30000

**结果**

[{  
 "id": "WakefieldFamily"  
}]

如果在你的索引策略中包含空间索引，则将通过索引有效地进行“距离查询”。有关空间索引的更多详细信息，请参阅以下章节。如果你没有指定路径的空间索引，仍然可以通过指定 x-ms-documentdb-query-enable-scan 请求标头（其值设置为“true”）执行空间查询。在 .NET 中，可以通过将可选的 **FeedOptions** 参数传递到 [EnableScanInQuery](https://msdn.microsoft.com/library/microsoft.azure.documents.client.feedoptions.enablescaninquery.aspx#P:Microsoft.Azure.Documents.Client.FeedOptions.EnableScanInQuery) 设置为 true 的查询来完成此操作。

ST\_WITHIN 可用于检查点是否在多边形内。多边形通常用来表示边界，例如邮政编码、省/自治区边界或自然构成物。再次说明，如果在你的索引策略中包含空间索引，则将通过索引有效地进行“within”查询。

ST\_WITHIN 中的多边形参数只能包含单个环形，也就是说，多边形本身不能包含洞。有关 ST\_WITHIN 查询的多边形中允许的点数目上限，请查看 [DocumentDB 限制](/documentation/articles/documentdb-limits)。

**查询**

SELECT \*   
FROM Families f   
WHERE ST\_WITHIN(f.location, {  
 'type':'Polygon',   
 'coordinates': [[[31.8, -5], [32, -5], [32, -4.7], [31.8, -4.7], [31.8, -5]]]  
})

**结果**

[{  
 "id": "WakefieldFamily",  
}]

[AZURE.NOTE] 与 DocumentDB 查询中不匹配类型的工作方式类似，如果任一参数中指定的位置值格式不正确或无效，则会评估为**未定义**，并且会在查询结果中跳过已评估的文档。如果你的查询没有返回任何结果，请运行 ST\_ISVALIDDETAILED 进行调试，以了解空间类型无效的原因。

ST\_ISVALID 和 ST\_ISVALIDDETAILED 可用来检查空间对象是否有效。例如，下列查询检查纬度值 (-132.8) 超出范围的点的有效性。ST\_ISVALID 仅返回一个布尔值，ST\_ISVALIDDETAILED 则返回布尔值和字符串，字符串中包含被视为无效的原因。

**查询**

SELECT ST\_ISVALID({ "type": "Point", "coordinates": [31.9, -132.8] })

**结果**

[{  
 "$1": false  
}]

这些函数也可以用来验证多边形。例如，在这里我们使用 ST\_ISVALIDDETAILED 来验证未闭合的多边形。

**查询**

SELECT ST\_ISVALIDDETAILED({ "type": "Polygon", "coordinates": [[   
 [ 31.8, -5 ], [ 31.8, -4.7 ], [ 32, -4.7 ], [ 32, -5 ]   
 ]]})

**结果**

[{  
 "$1": {   
 "valid": false,   
 "reason": "The Polygon input is not valid because the start and end points of the ring number 1 are not the same. Each ring of a polygon must have the same start and end points."   
 }  
}]

### .NET SDK 中的 LINQ 查询

DocumentDB .NET SDK 还提供存根方法 Distance() 和 Within()，供你在 LINQ 表达式中使用。DocumentDB LINQ 提供程序会将这些方法调用转换为等效的 SQL 内置函数调用（分别为 ST\_DISTANCE 和 ST\_WITHIN）。

以下是 LINQ 查询的示例，该查询使用 LINQ 在 DocumentDB 集合中查找所有文档，这些文档的“位置”值在指定点的 30 公里半径内。

**LINQ 距离查询**

foreach (UserProfile user in client.CreateDocumentQuery<UserProfile>(UriFactory.CreateDocumentCollectionUri("db", "profiles"))  
 .Where(u => u.ProfileType == "Public" && a.Location.Distance(new Point(32.33, -4.66)) < 30000))  
{  
 Console.WriteLine("\t" + user);  
}

同样地，以下查询查找所有文档，这些文档的“位置”均在指定的方框/多边形内。

**LINQ Within 查询**

Polygon rectangularArea = new Polygon(  
 new[]  
 {  
 new LinearRing(new [] {  
 new Position(31.8, -5),  
 new Position(32, -5),  
 new Position(32, -4.7),  
 new Position(31.8, -4.7),  
 new Position(31.8, -5)  
 })  
 });  
  
foreach (UserProfile user in client.CreateDocumentQuery<UserProfile>(UriFactory.CreateDocumentCollectionUri("db", "profiles"))  
 .Where(a => a.Location.Within(rectangularArea)))  
{  
 Console.WriteLine("\t" + user);  
}

我们已经探讨过如何使用 LINQ 和 SQL 查询文档，现在我们来看一下如何针对空间索引配置 DocumentDB。

## 索引

如[使用 Azure DocumentDB 进行架构不可知的索引](http://www.vldb.org/pvldb/vol8/p1668-shukla.pdf)一文中所述，我们设计的 DocumentDB 数据库引擎具有真正不可知的架构，并提供一流的 JSON 支持。DocumentDB 的写入优化数据库引擎现在也可以通过本机方式了解用 GeoJSON 标准表示的空间数据。

简单来说，测地坐标的几何图形会投影在 2D 平面上，然后使用 **quadtree** 以渐进方式划分成格子。这些格子会根据 **Hilbert 空间填充曲线**内的格子位置映射到 1D，并保留点的位置。此外，当位置数据进行索引编制后，会经历称为**分割**的过程，也就是说，在某个位置上相交的所有格子都会被识别为键并存储在 DocumentDB 索引中。在查询时，点和多边形等参数也会经过分割，以提取相关的格子 ID 范围，然后用于从索引检索数据。

如果指定的索引策略包含 /\*（所有路径）的空间索引，则表示在集合中找到的所有点均已编制索引，能进行有效的空间查询（ST\_WITHIN 和 ST\_DISTANCE）。空间索引没有精度值，并且始终使用默认的精度值。

以下 JSON 代码段演示已启用空间索引的索引策略，也就是为文档中找到的所有 GeoJSON 点编制索引，以用于空间查询。如果你要使用 Azure 门户修改索引策略，可以为索引策略指定以下 JSON，以便对集合启用空间索引。

**已启用空间的集合索引策略 JSON**

{  
 "automatic":true,  
 "indexingMode":"Consistent",  
 "includedPaths":[  
 {  
 "path":"/\*",  
 "indexes":[  
 {  
 "kind":"Range",  
 "dataType":"String",  
 "precision":-1  
 },  
 {  
 "kind":"Range",  
 "dataType":"Number",  
 "precision":-1  
 },  
 {  
 "kind":"Spatial",  
 "dataType":"Point"  
 }  
 ]  
 }  
 ],  
 "excludedPaths":[  
 ]  
}

以下是 .NET 中的代码段，演示如何创建针对所有包含点的路径启用空间索引的集合。

**创建具有空间索引的集合**

DocumentCollection spatialData = new DocumentCollection()  
spatialData.IndexingPolicy = new IndexingPolicy(new SpatialIndex(DataType.Point)); //override to turn spatial on by default  
collection = await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), spatialData);

下面说明了如何修改现有的集合，以便对文档内存储的所有点使用空间索引。

**修改具有空间索引的现有集合**

Console.WriteLine("Updating collection with spatial indexing enabled in indexing policy...");  
collection.IndexingPolicy = new IndexingPolicy(new SpatialIndex(DataType.Point));  
await client.ReplaceDocumentCollectionAsync(collection);  
  
Console.WriteLine("Waiting for indexing to complete...");  
long indexTransformationProgress = 0;  
while (indexTransformationProgress < 100)  
{  
 ResourceResponse<DocumentCollection> response = await client.ReadDocumentCollectionAsync(UriFactory.CreateDocumentCollectionUri("db", "coll"));  
 indexTransformationProgress = response.IndexTransformationProgress;  
  
 await Task.Delay(TimeSpan.FromSeconds(1));  
}

[AZURE.NOTE] 如果文档中的 GeoJSON 位置值格式不正确或无效，则不会为其编制索引以用于空间查询。你可以使用 ST\_ISVALID 和 ST\_ISVALIDDETAILED 验证位置值。

如果你的集合定义包含分区键，则不会报告索引转换进度。

## 后续步骤

你已经学会如何开始使用 DocumentDB 中的地理空间支持，现在可以：

* 使用 [Github 上的地理空间 .NET 代码示例](https://github.com/Azure/azure-documentdb-dotnet/blob/e880a71bc03c9af249352cfa12997b51853f47e5/samples/code-samples/Geospatial/Program.cs)开始编写代码
* 在 [DocumentDB 查询板块](http://www.documentdb.com/sql/demo#geospatial)中实际操作地理空间查询
* 详细了解 [DocumentDB 查询](/documentation/articles/documentdb-sql-query)
* 详细了解 [DocumentDB 索引策略](/documentation/articles/documentdb-indexing-policies)