# DocumentDB 索引策略

尽管许多客户都愿意让 DocumentDB 自动处理[索引的方方面面](documentdb-indexing.md)，但 DocumentDB 还支持在创建过程中为集合指定自定义**索引策略**。与其他数据库平台中提供的辅助索引相比，DocumentDB 中的索引策略更加灵活且功能强大，因为它允许你设计和自定义索引形状，而无需牺牲架构的灵活性。若要了解 DocumentDB 中的索引工作原理，就必须通过管理索引策略来了解它，你可以在索引存储开销、写入和查询吞吐量以及查询一致性之间进行详细权衡。

在本文中，我们将仔细研究 DocumentDB 索引策略、自定义索引策略的方法和相关的权衡方案。

阅读本文之后，你将能够回答以下问题：

* 如何重写属性以包括在索引中或从索引中排除？
* 如何为最终更新配置索引？
* 如何配置索引以执行 Order By 或范围查询？
* 如何对集合的索引策略进行更改？
* 如何比较不同索引策略的存储和性能？

## 自定义集合的索引策略

通过重写 DocumentDB 集合的默认索引策略并配置以下几个方面，开发人员可以在存储、写入/查询性能和查询一致性之间进行权衡。

* **包括在索引中/从索引中排除文档和路径**。开发人员可以选择在向集合中插入文档或替换文档时，要从索引中排除或包括在索引中的某些文档。开发人员还可以针对某个索引中包含的文档，选择在索引时包括或排除某些 JSON 属性，也称为路径（包括通配符模式）。
* **配置各种索引类型**。对于每个包括的路径，开发人员还可以根据其数据和预期查询工作负荷以及每个路径的数字/字符串“精度”，指定集合上需要的索引类型。
* **配置索引更新模式**。DocumentDB 支持三种索引模式，可通过索引策略对 DocumentDB 集合进行配置︰一致、延迟和无。

下面的.NET 代码段演示了如何在集合创建过程中设置自定义索引策略。这里我们以最大精度为字符串和数字设置范围索引策略。此策略可以让我们对字符串执行 Order By 查询。

DocumentCollection collection = new DocumentCollection { Id = "myCollection" };  
  
collection.IndexingPolicy = new IndexingPolicy(new RangeIndex(DataType.String) { Precision = -1 });  
collection.IndexingPolicy.IndexingMode = IndexingMode.Consistent;  
  
await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), collection);

[AZURE.NOTE] REST API 2015-06-03 版本更改了索引策略的 JSON 架构，以支持针对字符串的范围索引。.NET SDK 1.2.0 和 Java、Python 和 Node.js SDK 1.1.0 支持新策略架构。旧 SDK 使用 REST API 2015-04-08 版本，支持旧的索引策略架构。

默认情况下，DocumentDB 总是使用哈希索引对文档中的所有字符串属性执行索引，并使用范围索引对数值属性执行索引。

### 数据库索引模式

DocumentDB 支持三种索引模式，可通过索引策略对 DocumentDB 集合进行配置︰一致、延迟和无。

**一致**︰如果将 DocumentDB 集合的策略指定为“一致”，针对指定 DocumentDB 集合的查询将按照为点读取指定的一致性级别进行（即强、有限过期性、会话或最终）。索引会作为文档更新（即插入、替换、更新和删除 DocumentDB 集合中的文档）的一部分进行同步更新。一致索引支持一致的查询，但代价是可能降低写入吞吐量。这种减少受需要索引的唯一路径和“一致性级别”的影响。一致的索引模式适用于“快速写入、立即查询”的工作负荷。

**延迟**︰若要实现最大的文档引入吞吐量，可为 DocumentDB 集合配置延迟一致性；也就是说，查询最终是一致的。DocumentDB 集合处于静止状态时（即为用户请求提供服务时没有完全利用集合的吞吐量），将以异步方式更新索引。对于需要无阻碍文档引入的“现在引入、稍后查询”工作负荷，可能适合“延迟”索引模式。

**无**︰索引模式标记为“无”的集合没有与之关联的索引。如果 DocumentDB 用作键值存储，并且只通过其 ID 属性访问文档，则通常使用该模式。

[AZURE.NOTE] 将索引策略配置为“无”时，删除任何现有索引会产生不良影响。如果你的访问模式只需要 ID 和/或“自助链接”，请使用此选项。

下面的示例演示了如何使用 .NET SDK 借助针对所有文档插入的一致自动索引创建 DocumentDB 集合。

下表显示了根据为集合配置的索引模式（一致和延迟）和为查询请求指定的一致性级别进行查询的一致性。这适用于使用任何接口（REST API、SDK 或在存储过程和触发器中）进行的查询。

<tbody>  
 <tr>  
 <td valign="top">  
 <p>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 <strong>一致</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 <strong>延迟</strong>  
 </p>  
 </td>   
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 <strong>强</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 强  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 <strong>有限过期性</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 有限过期性  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 <strong>会话</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 会话  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 <strong>最终</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 </tr>   
</tbody>

DocumentDB 返回在“无”索引模式下对集合进行查询的错误。查询仍然通过 REST API 中的显式 x-ms-documentdb-enable-scans 标头或使用.NET SDK 的 EnableScanInQuery 请求选项来执行扫描。EnableScanInQuery 扫描不支持诸如 ORDER BY 等查询功能。

下表显示了指定 EnableScanInQuery 时，根据索引模式（一致、延迟和无）进行查询的一致性。

<tbody>  
 <tr>  
 <td valign="top">  
 <p>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 <strong>一致</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 <strong>延迟</strong>  
 </p>  
 </td>   
 <td valign="top">  
 <p>  
 <strong>无</strong>  
 </p>  
 </td>   
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 <strong>强</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 强  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 <td valign="top">  
 <p>  
 强  
 </p>  
 </td>   
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 <strong>有限过期性</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 有限过期性  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 <td valign="top">  
 <p>  
 有限过期性  
 </p>  
 </td>   
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 <strong>会话</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 会话  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 <td valign="top">  
 <p>  
 会话  
 </p>  
 </td>   
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 <strong>最终</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 <td valign="top">  
 <p>  
 最终  
 </p>  
 </td>   
 </tr>   
</tbody>

下面的代码示例演示如何使用.NET SDK 借助针对所有文档插入的一致自动索引创建 DocumentDB 集合。

// Default collection creates a hash index for all string and numeric   
 // fields. Hash indexes are compact and offer efficient  
 // performance for equality queries.  
   
 var collection = new DocumentCollection { Id ="defaultCollection" };  
   
 collection.IndexingPolicy.IndexingMode = IndexingMode.Consistent;  
   
 collection = await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("mydb"), collection);

### 索引路径

DocumentDB 将 JSON 文档和索引建立为树形，从而可以针对树中的路径调整策略。你可以在本 [DocumentDB 索引介绍](/documentation/articles/documentdb-indexing)中找到更多详细信息。在这些文档中，你可以选择必须包括在索引中或从索引中排除的路径。如果事先已知查询模式，这可以提高写入性能并减少方案所需的索引存储。

索引路径以根 (/) 开头，通常以通配符“?”结尾，表示前缀有多个可能值。例如，对于 SELECT \* FROM Families F WHERE F.familyName = “Andersen”，必须在集合的索引策略中包含 /familyName/? 的索引路径。

索引路径还可以使用 \* 通配符以递归方式指定路径在前缀下的行为。例如，/payload/\* 可用于从索引中排除负载属性下的所有内容。

下面是用于指定索引路径的常用模式︰

<tbody>  
 <tr>  
 <td valign="top">  
 <p>  
 <strong>路径</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 <strong>说明/用例</strong>  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 /\*  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 集合的默认路径。递归并应用于整个文档树。  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 /prop/?  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 执行查询所需的索引路径（分别为哈希或范围类型）︰  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop = "value"  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop > 5  
 </p>  
 <p>  
 SELECT \* FROM collection c ORDER BY c.prop  
 </p>   
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 /prop/\*  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 指定标签下所有路径的索引路径。使用以下查询  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop = "value"  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop.subprop > 5  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop.subprop.nextprop = "value"  
 </p>  
 <p>  
 SELECT \* FROM collection c ORDER BY c.prop  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 /props/[]/?  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 对标量数组（如 ["a"、"b"、"c"]）进行迭代和 JOIN 查询所需的索引路径︰  
 </p>  
 <p>  
 SELECT tag FROM tag IN collection.props WHERE tag = "value"  
 </p>  
 <p>  
 SELECT tag FROM collection c JOIN tag IN c.props WHERE tag > 5  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 /props/[]/subprop/?  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 针对诸如 [{subprop:"a"}, {subprop:"b"}] 的对象数组进行迭代和 JOIN 查询所需的索引路径：  
 </p>  
 <p>  
 SELECT tag FROM tag IN collection.props WHERE tag.subprop = "value"  
 </p>  
 <p>  
 SELECT tag FROM collection c JOIN tag IN c.props WHERE tag.subprop = "value"  
 </p>  
 </td>  
 </tr>   
 <tr>  
 <td valign="top">  
 <p>  
 /prop/subprop/?  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 进行查询所需的索引路径 （分别为哈希或范围类型）︰  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop.subprop = "value"  
 </p>  
 <p>  
 SELECT \* FROM collection c WHERE c.prop.subprop > 5  
 </p>  
 <p>  
 SELECT \* FROM collection c ORDER BY c.prop.subprop  
 </p>   
 </td>  
 </tr>  
</tbody>

[AZURE.NOTE] 在设置自定义索引路径时，需要为由特殊路径“/\*”表示的整个文档树指定默认索引规则。

下面的示例通过范围索引和 20 个字节的自定义精度值来配置特定路径：

var collection = new DocumentCollection { Id = "rangeSinglePathCollection" };   
  
collection.IndexingPolicy.IncludedPaths.Add(  
 new IncludedPath {   
 Path = "/Title/?",   
 Indexes = new Collection<Index> {   
 new RangeIndex(DataType.String) { Precision = 20 } }   
 });  
  
// Default for everything else  
collection.IndexingPolicy.IncludedPaths.Add(  
 new IncludedPath {   
 Path = "/\*" ,  
 Indexes = new Collection<Index> {  
 new HashIndex(DataType.String) { Precision = 3 },   
 new RangeIndex(DataType.Number) { Precision = -1 }   
 }  
 });  
   
collection = await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), pathRange);

### 索引数据类型、种类和精度

现在，我们已经介绍了如何指定路径，让我们来讨论配置路径的索引策略时可以使用的选项。可以为每个路径指定一个或多个索引定义︰

* 数据类型︰ **字符串**、**数值**或**点**（每个路径每个数据只可包含一个条目）
* 索引类型︰**哈希**（等式查询）、**范围**（等式、范围或 Order By 查询）或**空间**（空间查询）
* 精度︰对于数字为 1-8 或 -1（最大精度），对于字符串为 1-100（最大精度）

#### 索引种类

DocumentDB 针对每个路径都支持哈希和范围索引类型（即可以配置为字符串、数值或两者）。

* **哈希**支持高效的等式查询和 JOIN 查询。在大多数使用情况下，哈希索引需要的精度不会高于 3 个字节的默认值。
* **范围**支持高效的等式查询、范围查询（分别使用 >、<、>=、<=、!=）和 Order By 查询。默认情况下，Order By 查询还需要最大索引精度 (-1)。

DocumentDB 还针对每个路径支持空间索引，可为点数据类型指定空间索引。指定路径中的值必须是有效的 GeoJSON 点，如 {"type": "Point", "coordinates": [0.0, 10.0]}。

* **空间**支持高效的空间（在其中和距离）查询。

下面是支持的索引类型以及可以使用的查询示例︰

<tbody>  
 <tr>  
 <td valign="top">  
 <p>  
 <strong>索引种类</strong>  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 <strong>说明/用例</strong>  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 哈希  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 通过 /prop /?（或 / \*）的哈希可用于高效执行以下查询：SELECT \* FROM collection c WHERE c.prop = "value"。通过 /props/[]/?（或 /\*或 /props/\*）的哈希可用于高效执行以下查询：SELECT tag FROM collection c JOIN tag IN c.props WHERE tag = 5  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 范围  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 通过 /prop /?（或 / \*）的范围可用于高效执行以下查询：SELECT \* FROM collection c WHERE c.prop = "value" SELECT \* FROM collection c WHERE c.prop > 5 SELECT \* FROM collection c ORDER BY c.prop  
 </p>  
 </td>  
 </tr>  
 <tr>  
 <td valign="top">  
 <p>  
 空间  
 </p>  
 </td>  
 <td valign="top">  
 <p>  
 通过 /prop /?（或 / \*）的范围可用于高效执行以下查询：SELECT \* FROM collection c WHERE ST\_DISTANCE(c.prop, {"type": "Point", "coordinates": [0.0, 10.0]}) &lt; 40 SELECT \* FROM collection c WHERE ST\_WITHIN(c.prop, {"type": "Polygon", ... })  
 </p>  
 </td>  
 </tr>   
</tbody>

默认情况下，如果没有范围索引（任何精度），则使用范围运算符（如 > =）的查询会返回错误，以提示执行查询必须执行一次扫描。使用 REST API 中的 x-ms-documentdb-enable-scans 标头或使用.NET SDK 的 EnableScanInQuery 请求选项，可以在没有范围索引的情况下执行范围查询。如果在 DocumentDB 可以使用索引来筛选的查询中有其他筛选器，则会不返回错误。

空间查询的规则相同。默认情况下，如果没有空间索引，并且索引中没有其他筛选器可以使用，则空间查询会返回错误。可以使用 x-ms-documentdb-enable-scan/EnableScanInQuery 以扫描方式执行空间索引。

#### 索引精度

索引精度让你可以在索引存储开销和查询性能之间做出权衡。对于数值，我们建议使用默认的精度配置 -1（最大）。由于数字是 JSON 格式的 8 个字节，这相当于 8 个字节的配置。选择较低值的精度（如 1-7）意味着在某些范围内的值会映射到相同的索引条目。因此，你可以降低索引存储空间，但查询执行可能需要处理更多文档，并因此占用更大的吞吐量，即请求单位。

索引精度配置对于字符串范围更加实用。由于字符串可以是任意长度，索引精度的选择可影响字符串范围查询的性能，并且影响所需的索引存储空间量。字符串范围索引可以配置为 1-100 或 -1（最大）。如果想要对字符串属性执行 Order By 查询，则必须为相应路径指定精度 -1。

空间索引始终为点使用默认索引精度，并且不能重写。

下面的示例演示了如何使用 .NET SDK 来提高集合中范围索引的精度。

**创建自定义索引精度的集合**

var rangeDefault = new DocumentCollection { Id = "rangeCollection" };  
  
// Override the default policy for Strings to range indexing and "max" (-1) precision  
rangeDefault.IndexingPolicy = new IndexingPolicy(new RangeIndex(DataType.String) { Precision = -1 });  
  
await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), rangeDefault);

[AZURE.NOTE] 当查询使用 Order By，但针对最大精度的查询路径没有范围索引时，DocumentDB 将返回错误。

同样，可以从索引中完全排除路径。下面的示例演示了如何使用“\*”通配符从索引中排除文档的整个部分（也称为子树）。

var collection = new DocumentCollection { Id = "excludedPathCollection" };  
collection.IndexingPolicy.IncludedPaths.Add(new IncludedPath { Path = "/\*" });  
collection.IndexingPolicy.ExcludedPaths.Add(new ExcludedPath { Path = "/nonIndexedContent/\*");  
  
collection = await client.CreateDocumentCollectionAsync(UriFactory.CreateDatabaseUri("db"), excluded);

## 选择包括在索引中和从索引中排除

可以选择是否让集合自动为所有文档执行索引。默认情况下，为所有文档自动执行索引，但你可以选择关闭该功能。关闭索引功能后，只能通过本身的链接或通过使用 ID 进行查询的方法访问文档。

关闭自动索引后，你仍然可以选择性地只将特定的文档添加到索引中。相反，可以保留自动索引，并选择只排除特定的文档。当只需要查询一个文档子集时，索引开/关配置非常有用。

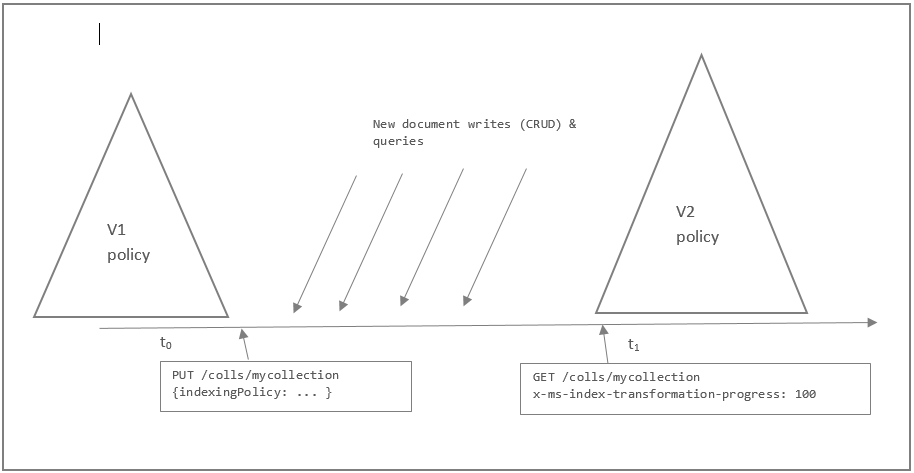
例如，下面的示例演示了如何使用 [DocumentDB.NET SDK](https://github.com/Azure/azure-documentdb-java) 和 [RequestOptions.IndexingDirective](http://msdn.microsoft.com/library/microsoft.azure.documents.client.requestoptions.indexingdirective.aspx) 属性来显式包括文档。

// If you want to override the default collection behavior to either  
// exclude (or include) a Document from indexing,  
// use the RequestOptions.IndexingDirective property.  
client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri("db", "coll"),  
 new { id = "AndersenFamily", isRegistered = true },  
 new RequestOptions { IndexingDirective = IndexingDirective.Include });

## 修改集合的索引策略

DocumentDB 允许你动态更改集合的索引策略。更改 DocumentDB 集合的索引策略可能导致索引形状改变，包括可编制索引的路径、其精度以及索引本身的一致性模型。因此，索引策略的更改实际上要求将旧索引转换为新索引。

**联机索引转换**



索引的工作原理 – DocumentDB 联机索引转换

在联机状态下执行索引转换，这意味着按照旧策略索引的文档可以按照新策略有效转换，**而不会影响集合的写入可用性或配置的吞吐量**。在索引转换过程中，使用 REST API、SDK 或在存储的过程和触发器中执行读取和写入操作的一致性不会受到影响。这就意味着，在更改索引策略时，你的应用程序的性能不会下降，也没有停机时间。

但是，无论索引模式配置如何（一致或延迟），在执行索引转换期间查询始终一致。这适用于使用所有接口（REST API、SDK 或从存储过程和触发器中）进行的查询。与延迟索引一样，使用特定副本可用的备用资源在后台以异步方式对副本执行索引转换。

索引转换还会**就地**进行，即 DocumentDB 不维护索引的两个副本，而是用新索引代替旧索引。这就意味着，执行索引转换时集合中不需要也不占用额外的磁盘空间。

更改索引策略时，与其他值（如包括/排除的路径、索引类型和精度）相比，如何应用更改将旧索引转换为新索引主要取决于索引模式配置。如果新旧策略都使用一致的索引，则 DocumentDB 执行联机索引转换。进行转换时，不能在一致索引模式下应用另一个索引策略应用更改。

但是在进行转换时，可以切换到延迟或无索引模式。

* 当切换到延迟模式时，索引策略更改立即生效，并且 DocumentDB 开始以异步方式重新创建索引。
* 当切换到无模式时，索引会被立即删除。当想要取消正在进行的转换，并开始使用不同的索引策略时，切换到无模式非常有用。

如果使用 .NET SDK，则可以使用新的 **ReplaceDocumentCollectionAsync** 方法进行索引策略更改，并利用从 **ReadDocumentCollectionAsync** 调用中获得的 **IndexTransformationProgress** 响应属性跟踪索引转换的进度百分比。其他 SDK 和 REST API 支持使用等效属性和方法进行索引策略更改。

以下代码段演示了如何将集合的索引策略从一致索引模式改为延迟。

**将索引策略从一致改为延迟**

// Switch to lazy indexing.  
Console.WriteLine("Changing from Default to Lazy IndexingMode.");  
  
collection.IndexingPolicy.IndexingMode = IndexingMode.Lazy;  
  
await client.ReplaceDocumentCollectionAsync(collection);

可以通过调用 ReadDocumentCollectionAsync 来查看索引转换的进度，例如下图所示。

**跟踪索引转换进度**

long smallWaitTimeMilliseconds = 1000;  
long progress = 0;  
  
while (progress < 100)  
{  
 ResourceResponse<DocumentCollection> collectionReadResponse = await client.ReadDocumentCollectionAsync(  
 UriFactory.CreateDocumentCollectionUri("db", "coll"));  
  
 progress = collectionReadResponse.IndexTransformationProgress;  
  
 await Task.Delay(TimeSpan.FromMilliseconds(smallWaitTimeMilliseconds));  
}

通过切换到无索引模式，可以删除集合的索引。如果想要取消正在进行的转换并立即启动一个新的索引，这个操作工具可能非常有用。

**删除集合的索引**

// Switch to lazy indexing.  
Console.WriteLine("Dropping index by changing to to the None IndexingMode.");  
  
collection.IndexingPolicy.IndexingMode = IndexingMode.None;  
  
await client.ReplaceDocumentCollectionAsync(collection);

你何时将更改 DocumentDB 集合的索引策略？ 以下使用案例最常见：

* 在正常操作中提供一致的结果，但在批量数据导入时返回到延迟索引
* 对当前 DocumentDB 集合开始使用新索引功能，例如需要空间索引类型的地理空间查询，或需要字符串范围索引类型的 Order By/字符串范围查询
* 手动选择要编制索引的属性，并随着时间的推移进行更改
* 调整索引精度，以提高查询性能或减少占用的存储

[AZURE.NOTE] 若要使用 ReplaceDocumentCollectionAsync 修改索引策略，要求安装不低于 1.3.0 版本的 .NET SDK。

要成功完成索引转换，必须确保集合有足够的可用存储空间。如果集合达到其存储配额，将暂停索引转换。获得可用的存储空间后（例如删除某些文档），索引转换将立即自动恢复。

## 性能调优

DocumentDB API 提供有关性能指标的信息，如所用的索引存储以及每次操作的吞吐量成本（请求单位）。此信息可用于比较各种索引策略和优化性能。

若要检查存储配额和集合用法，请针对集合资源运行 HEAD 或 GET 请求，并检查 x-ms-request-quota 和 x-ms-request-usage 标头。在 .NET SDK 中，[ResourceResponse](http://msdn.microsoft.com/library/dn799209.aspx) 中的 [DocumentSizeQuota](http://msdn.microsoft.com/library/dn850325.aspx) 和 [DocumentSizeUsage](http://msdn.microsoft.com/library/azure/dn850324.aspx) 属性包含这些相应的值。

// Measure the document size usage (which includes the index size) against   
 // different policies.  
 ResourceResponse<DocumentCollection> collectionInfo = await client.ReadDocumentCollectionAsync(UriFactory.CreateDocumentCollectionUri("db", "coll"));   
 Console.WriteLine("Document size quota: {0}, usage: {1}", collectionInfo.DocumentQuota, collectionInfo.DocumentUsage);

要测量每次写入操作（创建、更新或删除）执行索引的开销，请检查 x-ms-request-charge header 标头（或同等的 .NET SDK 中 [ResourceResponse < T >](http://msdn.microsoft.com/library/dn799209.aspx) 中的 [RequestCharge](http://msdn.microsoft.com/library/dn799099.aspx) 属性）来测量这些操作占用的请求单位数。

// Measure the performance (request units) of writes.   
 ResourceResponse<Document> response = await client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri("db", "coll"), myDocument);   
 Console.WriteLine("Insert of document consumed {0} request units", response.RequestCharge);  
   
 // Measure the performance (request units) of queries.   
 IDocumentQuery<dynamic> queryable = client.CreateDocumentQuery(UriFactory.CreateDocumentCollectionUri("db", "coll"), queryString).AsDocumentQuery();  
  
 double totalRequestCharge = 0;  
 while (queryable.HasMoreResults)  
 {  
 FeedResponse<dynamic> queryResponse = await queryable.ExecuteNextAsync<dynamic>();   
 Console.WriteLine("Query batch consumed {0} request units",queryResponse.RequestCharge);  
 totalRequestCharge += queryResponse.RequestCharge;  
 }  
   
 Console.WriteLine("Query consumed {0} request units in total", totalRequestCharge);

## 对索引策略规范的更改

在 REST API 2015-06-03 版本中，于 2015 年 7 月 7 日对索引策略的架构进行了更改。SDK 版本中的相应类具有与架构匹配的新实现。

JSON 规范中实现了以下更改︰

* 索引策略针对字符串支持范围索引
* 每个路径可以有多个索引定义，每个对应一种数据类型
* 对于数字，索引精度支持 1-8，对于字符串支持 1-100，-1 为最大精度。
* 路径段不需要双引号来转义每个路径。例如，你可以添加 /title/? 路径，而不是 /“title”/?
* 表示“所有路径”的根路径可以表示为 /\*（除了 /）

如果你的代码使用 1.1.0 或更高版本的 .NET SDK 编写的自定义索引策略设置集合，则需要更改应用程序代码来处理这些更改，以迁移到 SDK 1.2.0 版本。如果你没有用于配置索引策略的代码，或打算继续使用 SDK 旧版本，则不需进行任何更改。

为了进行实际比较，下面给出一个自定义索引策略示例，该策略是用 REST API 2015-06-03 版本和早期的 2015-04-08 版本编写的。

**以前的索引策略 JSON**

{  
 "automatic":true,  
 "indexingMode":"Consistent",  
 "IncludedPaths":[  
 {  
 "IndexType":"Hash",  
 "Path":"/",  
 "NumericPrecision":7,  
 "StringPrecision":3  
 }  
 ],  
 "ExcludedPaths":[  
 "/"nonIndexedContent"/\*"  
 ]  
}

**当前的索引策略 JSON**

{  
 "automatic":true,  
 "indexingMode":"Consistent",  
 "includedPaths":[  
 {  
 "path":"/\*",  
 "indexes":[  
 {  
 "kind":"Hash",  
 "dataType":"String",  
 "precision":3  
 },  
 {  
 "kind":"Hash",  
 "dataType":"Number",  
 "precision":7  
 }  
 ]  
 }  
 ],  
 "ExcludedPaths":[  
 {  
 "path":"/nonIndexedContent/\*"  
 }  
 ]  
}

## 后续步骤

通过下面的链接查看索引策略管理示例，并了解有关 DocumentDB 的查询语言的详细信息。

1. [DocumentDB .NET 索引管理代码示例](https://github.com/Azure/azure-documentdb-net/blob/master/samples/code-samples/IndexManagement/Program.cs)
2. [DocumentDB REST API 集合操作](https://msdn.microsoft.com/library/azure/dn782195.aspx)
3. [使用 DocumentDB SQL 进行查询](/documentation/articles/documentdb-sql-query)