# DocumentDB 分层资源模型和概念

DocumentDB 管理的数据库实体被称为**资源**。每个资源都通过逻辑 URI 进行唯一标识。你可以使用标准 HTTP 谓词、请求/响应标头和状态代码与资源进行交互。

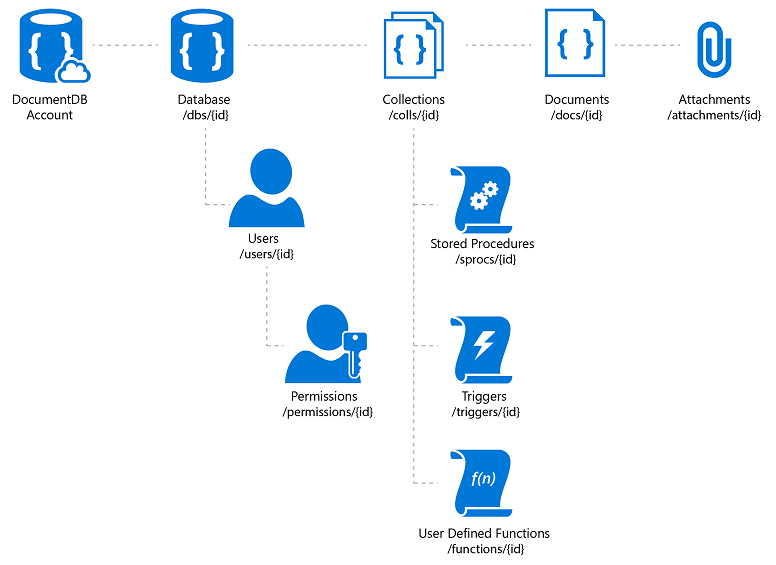
通过阅读本文，你将能够回答以下问题：

* DocumentDB 资源模型是什么？
* 对照用户定义的资源，什么是系统定义的资源？
* 如何对资源进行寻址？
* 如何使用集合？
* 如何使用存储过程、触发器和用户自定义函数 (UDF)？

## 分层资源模型

如下面的关系图所示，DocumentDB 分层**资源模型**由数据库帐户下的数组资源构成，每个帐户可通过一个逻辑且稳定的 URI 进行寻址。本文将一组资源称为一个**源**。

[AZURE.NOTE] DocumentDB 提供了高效的 TCP 协议，该协议在其通信模型中也是 RESTful，可通过 [.NET 客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx) 获得。

  
**分层资源模型**

若要开始使用资源，必须使用你的 Azure 订阅[创建 DocumentDB 数据库帐户](/documentation/articles/documentdb-create-account)。数据库帐户可以包含的一组**数据库**、每个数据库都包含多个**集合**，每个集合又包含**存储过程、触发器、UDF、文档**及相关**附件**（预览功能）。数据库也有关联的**用户**，每个用户都有一组**权限**，用于访问集合、存储过程、触发器、UDF、文档或附件。而数据库、用户、权限和集合就是系统定义的资源，其中已知的架构、文档和附件包含用户定义的任意 JSON 内容。

|  |  |
| --- | --- |
| 资源 | 说明 |
| 数据库帐户 | 每个数据库帐户都与一组数据库和一个固定大小的附件（预览功能）blob 存储相关联。你可以使用 Microsoft Azure 订阅创建一个或多个数据库帐户。有关详细信息，请访问[定价页](/pricing/details/documentdb/)。 |
| 数据库 | 数据库是跨集合分区的文档存储的逻辑容器。它也是一个用户容器。 |
| 用户 | 范围权限的逻辑命名空间。 |
| 权限 | 与用户关联用于访问特定资源的授权令牌。 |
| 集合 | 集合是 JSON 文档和相关联的 JavaScript 应用程序逻辑的容器。集合是一个计费实体，其中[成本](/documentation/articles/documentdb-performance-levels)由与集合关联的性能级别确定。集合可以跨一个或多个分区/服务器，并且能伸缩以处理几乎无限制增长的存储或吞吐量。 |
| 存储过程 | 以 JavaScript 编写的应用程序逻辑，使用集合注册且在数据库引擎中以事务方式执行。 |
| 触发器 | 在插入、替换或删除操作之前或之后执行的以 JavaScript 编写的应用程序逻辑。 |
| UDF | 用 JavaScript 编写的应用程序逻辑。UDF 让你可以建立自定义查询运算符模型，从而扩展核心 DocumentDB 查询语言。 |
| 文档 | 用户定义的（任意）JSON 内容。默认情况下，不需要定义任何架构，也不需要为所有添加到集合的文档提供辅助索引。 |
| （预览）附件 | 附件是一个特殊文档，包含引用和与外部 blob/媒体相关联的元数据。开发人员可以选择由 DocumentDB 来管理 blob 或者使用外部 blob 服务提供程序（OneDrive、Dropbox 等）来存储它。 |

## 系统定义的资源对比用户定义的资源

资源（例如数据库帐户、数据库、集合、用户、权限、存储过程、触发器和 UDF）都具有固定的架构并且都称为系统资源。与此相反，文档和附件这一类资源的架构不受限制，这一类资源就是用户定义的资源。在 DocumentDB 中，系统和用户定义的资源均由符合标准的 JSON 表示并进行管理。所有系统或用户定义的资源都具有以下公共属性。

[AZURE.NOTE] 请注意，资源中所有系统生成的属性在其 JSON 表示形式中前面都加有下划线 (\_)。

<tbody>  
 <tr>  
 <td valign="top"><p><strong>属性</strong></p></td>  
 <td valign="top"><p><strong>是用户设置的还是系统生成的？</strong></p></td>  
 <td valign="top"><p><strong>目的</strong></p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>\_rid</p></td>  
 <td valign="top"><p>系统生成的</p></td>  
 <td valign="top"><p>系统生成的、资源的唯一分层标识符</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>\_etag</p></td>  
 <td valign="top"><p>系统生成的</p></td>  
 <td valign="top"><p>乐观并发控制所需的资源的 ETag</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>\_ts</p></td>  
 <td valign="top"><p>系统生成的</p></td>  
 <td valign="top"><p>资源上次更新的时间戳</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>\_self</p></td>  
 <td valign="top"><p>系统生成的</p></td>  
 <td valign="top"><p>资源的唯一可寻址 URI</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>id</p></td>  
 <td valign="top"><p>系统生成的</p></td>  
 <td valign="top"><p>资源的用户定义的唯一名称（具有相同分区键值）。如果用户未指定 ID，系统将生成 ID</p></td>  
 </tr>  
</tbody>

### 资源的网络表示形式

DocumentDB 不强制对 JSON 标准或特殊编码进行任何专有扩展；它使用符合标准的 JSON 文档。

### 对资源进行寻址

所有资源都可通过 URI 寻址。**\_self** 属性的值表示资源的相对 URI。URI 的格式由 /<feed>/{\_rid} 路径段构成：

|\_self 的值 | 说明 |——————-|———– |/dbs |数据库帐户下的数据库的源 |/dbs/{dbName} |其 ID 与值 {dbName} 匹配的数据库 |/dbs/{dbName}/colls/ |数据库下的集合的源 |/dbs/{dbName}/colls/{collName} |其 ID 与值 {collName} 匹配的集合 |/dbs/{dbName}/colls/{collName}/docs |集合下的文档的源 |/dbs/{dbName}/colls/{collName}/docs/{docId} |其 ID 与值 {doc} 匹配的文档 |/dbs/{dbName}/users/ |数据库下的用户的源 |/dbs/{dbName}/users/{userId} |其 ID 与值 {user} 匹配的用户 |/dbs/{dbName}/users/{userId}/permissions |用户下的权限的源 |/dbs/{dbName}/users/{userId}/permissions/{permissionId} |其 ID 与值 {permission} 匹配的权限

每个资源都具有唯一的用户定义的名称，通过 ID 属性公开。请注意：对于文档，如果用户未指定 ID，系统将自动生成文档的唯一 ID。该 ID 是用户定义的字符串，长度为最多 256 个字符，且在特定父资源的上下文中是唯一的。

每个资源还具有系统生成的分层资源标识符（也称为 RID），可通过 \_rid 属性获得。RID 可对给定资源的整个层次结构进行编码，它是一种简便的内部表示形式，用于以分布式方式强制引用完整性。在数据库帐户内 RID 是唯一的，且由 DocumentDB 在内部使用，用于进行有效路由，而无需跨分区查找。\_self 和 \_rid 属性的值都是交替且规范的资源表示形式。

DocumentDB REST API 支持资源寻址和由 ID 和 \_rid 属性提出的请求路由。

## 数据库帐户

你可以使用 Azure 订阅创建一个或多个 DocumentDB 数据库帐户。将给予每个标准层数据库帐户最少一个 S1 集合的容量。

你可以通过在 [http://portal.azure.com/](https://portal.azure.cn/) 的 Azure 门户[创建和管理 DocumentDB 数据库帐户](/documentation/articles/documentdb-create-account)。创建和管理数据库帐户需要具有管理访问权限，并且只能在你的 Azure 订阅下执行。

### 数据库帐户属性

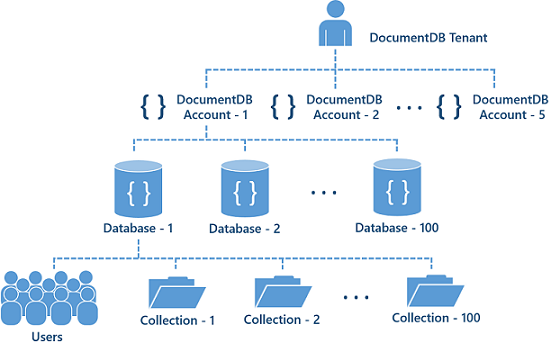
作为设置和管理数据库帐户的一部分，你可以配置并读取以下属性：

<tbody>  
 <tr>  
 <td valign="top"><p><strong>属性名称</strong></p></td>  
 <td valign="top"><p><strong>说明</strong></p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>一致性策略</p></td>  
 <td valign="top"><p>设置此属性来配置数据库帐户下的所有集合的默认一致性级别。你可以使用 [x-ms-consistency-level] 请求标头重写基于每个请求的一致性级别。<p><p>请注意，此属性仅适用于<i>用户定义的资源</i>。所有系统定义的资源都配置为支持具有高度一致性的读取/查询。</p></td>  
 </tr>  
 <tr>  
 <td valign="top"><p>授权密钥</p></td>  
 <td valign="top"><p>这些是提供对所有数据库帐户下的资源的管理访问权限的主要、次要和只读密钥。</p></td>  
 </tr>  
</tbody>

请注意，除了从 Azure 门户设置、配置和管理你的数据库帐户，你还可以通过使用 [Azure DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 和[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx)，以编程方式创建和管理 DocumentDB 数据库帐户。

## 数据库

DocumentDB 数据库是一个或多个集合和用户的逻辑容器，如下面的关系图中所示。你可以使用 DocumentDB 数据库帐户创建任意数量的数据库（取决于产品/服务限制）。

  
**数据库是用户和集合的逻辑容器**

数据库几乎可以包含由集合分区的无限文档存储，从而形成其中所包含的文档的事务域。

### DocumentDB 数据库的弹性缩放

DocumentDB 数据库默认情况下是弹性的 – 范围可以从几个 GB 到数千兆字节的 SSD 支持的文档存储和设置的吞吐量。

与传统的 RDBMS 中的数据库不同，DocumentDB 中的数据库范围不限于单台计算机。使用 DocumentDB，随着应用程序的缩放需求的不断增长，可以创建更多集合、数据库，或同时创建两者。实际上，在 Microsoft 内部，各种第一方应用程序已在以使用者规模使用 DocumentDB，其方式是通过创建非常大的 DocumentDB 数据库，每个数据库包含具有 TB 级别的文档存储的数千个集合。你可以通过添加或删除的集合来满足应用程序的缩放要求，从而扩大或收缩数据库。

你可以根据产品/服务在数据库内创建任意数量的集合。每个集合都具有 SSD 支持的存储和为你设置的吞吐量，具体取决于所选的性能层。

DocumentDB 数据库也是用户的容器。反过来，用户是一组权限的逻辑命名空间，可提供对集合、文档和附件的精细授权和访问权限。

与使用 DocumentDB 资源模型中的其他资源一样，可以使用 [Azure DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 或任一[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx) 轻松创建、替换、删除、读取或枚举数据库。DocumentDB 确保读取或查询数据库资源的元数据操作的高度一致性。自动删除数据库，确保你不能访问任何集合或它所包含的用户。

## 集合

DocumentDB 集合是 JSON 文档的一个容器。集合也是一组用于事务和查询的规模单位。

### SSD 支持的弹性文档存储

集合本质上是弹性的 - 当你添加或删除文档时，它会自动扩展或收缩。集合是逻辑资源，并且可以跨一个或多个物理分区或服务器。DocumentDB 基于存储大小和你的集合设置的吞吐量确定集合中的分区数。DocumentDB 中的每个分区都具有固定大小的与之关联的 SSD 支持的存储，并且会复制分区以实现高可用性。分区完全由 Azure DocumentDB 进行管理，因此，你无需编写复杂的代码或亲自管理分区。DocumentDB 集合在存储和吞吐量方面**几乎无限制**。

### 自动编制集合索引

DocumentDB 是真正无架构的数据库系统。无需为 JSON 文档假设或要求任何架构。将文档添加到集合时，DocumentDB 会自动编制索引，以供查询时使用。自动编制文档索引而无需架构或辅助索引是 DocumentDB 的关键功能，通过写优化、无锁定和日志结构化索引维护技术而实现。DocumentDB 支持持续大量的极快速写入，同时仍然保障一致的查询。文档和索引存储都可用于计算每个集合使用的存储。你可以控制存储和性能权衡，这些权衡与通过配置集合的索引编制策略进行的索引相关联。

### 配置集合的索引编制策略

每个集合的索引编制策略都可以使你将性能和存储权衡与编制索引关联起来。你可以使用以下选项进行索引编制配置：

* 选择是否让集合自动为所有文档编制索引。默认情况下，为所有文档自动编制索引。你可以选择关闭自动索引，并选择性地只将特定的文档添加到索引中。反过来，你也可以选择只排除特定的文档。可以通过将集合的 indexingPolicy 上的自动属性设置为 true 或 false，并在插入、替换或删除文档的同时使用 [x-ms-indexingdirective] 请求标头，从而实现此目的。
* 选择是否要在索引中包括特定的路径或文档中的模式或从索引中将其排除。你可以通过分别设置集合中的 indexingPolicy 上的 includedPaths 和 excludedPaths 来实现这一点。你还可以配置用于特定路径模式的存储和性能权衡的范围和哈希查询。
* 在同步（一致）和异步（延迟）索引更新之间进行选择。默认情况下，每次在集合中插入、替换或删除文档时同步更新索引。这个行为让查询能够使用与文档读取相同的一致性级别。虽然 DocumentDB 针对写入进行了优化，且支持文档持续写入，以及同步索引维护和提供一致的查询服务，但你也可以配置某些集合，使其索引延迟更新。延迟索引编制可大大提高写入性能，非常适合主要具有大量读取操作的集合的批量引入方案。

可以通过对集合执行 PUT 更改索引策略。这可以通过[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx)、[Azure 门户](https://portal.azure.com) 或 [Azure DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 来实现。

### 查询集合

集合中的文档可以具有任意的数据库架构，而你无需提前提供任何架构或辅助索引，就可以查询集合中的文档。你可以使用 [DocumentDB SQL 语法](https://msdn.microsoft.com/library/azure/dn782250.aspx)查询集合，该语法通过基于 JavaScript 的 UDF 提供丰富的分层、关系和空间运算符以及扩展性。JSON 语法允许将 JSON 文档建模为树，其中标签作为树节点。DocumentDB 的自动索引编制技术和 DocumentDB SQL 方言都利用了此语法。DocumentDB 查询语言包含三个主要方面：

1. 一小组查询操作，它自然映射到包括分层查询和投影的树结构。
2. 一小部分关系操作，包括组合、筛选、投影、聚合和自联接。
3. 基于纯 JavaScript 且结合 (1) 和 (2) 使用的 UDF。

DocumentDB 查询模型尝试在功能、效率和简单性之间取得平衡。DocumentDB 数据库引擎在本机上进行编译和执行 SQL 查询语句。你可以使用 [Azure DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 或任一[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx) 查询集合。.NET SDK 附带了 LINQ 提供程序。

[AZURE.TIP] 你可以尝试 DocumentDB 并在[查询版块](https://www.documentdb.com/sql/demo)中针对我们的数据集运行 SQL 查询。

### 多文档事务

数据库事务提供了一种安全且可预测的编程模型来处理数据的并发更改。在 RDBMS 中，编写业务逻辑的传统方法是编写**存储过程**和/或**触发器**，然后将它运到用于事务性执行的数据库服务器。在 RDBMS 中，需要应用程序编程人员来处理这两种不同的编程语言：

* （非事务性）应用程序编程语言（例如 JavaScript、Python、C#、Java 等。）
* 数据库在本机上执行的事务性编程语言 T-SQL

由于 DocumentDB 致力于直接在数据库引擎中使用 JavaScript 和 JSON，因此，在存储过程和触发器方面，它为在集合上直接执行基于 JavaScript 的应用程序逻辑提供了直观编程模型。这样就可以实现以下两点：

* 有效实现直接在数据库引擎中对 JSON 对象图进行并发控制、恢复、自动编制索引
* 直接使用 JavaScript 编程语言方面的数据库事务自然表达控制流、变量作用域、异常处理基元的分配和集成

在集合级别上注册的 JavaScript 逻辑稍后可以在给定集合的文档中发出数据库操作。DocumentDB 跨集合中的文档使用快照隔离在周围 ACID 事务中隐式地包装基于 JavaScript 的存储过程和触发器。在其执行过程中，如果 JavaScript 引发异常，则整个事务将被中止。生成的编程模型非常简单但功能强大。JavaScript 开发人员在继续使用其自己熟悉的语言构造和库基元的同时，可以获得“持久”的编程模型。

直接在与缓冲池位于相同地址空间内的数据库引擎中执行 JavaScript 的能力可实现针对集合的文档的数据库操作的高性能和事务性执行。此外，DocumentDB 数据库引擎还致力于针对 JSON 和 JavaScript 消除应用程序和数据库类型系统之间的任何阻抗失配。

创建集合之后即可使用 [Azure DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 或任一[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx) 为集合注册存储过程、触发器和 UDF。注册后，你可以引用并执行它们。请考虑完全以 JavaScript 编写的存储过程，下面的代码采用两个参数（书名和作者姓名）创建一个新文档，对文档进行查询后再对其进行更新 – 所有这一切都是在一个隐式的 ACID 事务内完成。在执行期间的任何时刻，如果引发 JavaScript 异常，则中止整个事务。

function businessLogic(name, author) {  
 var context = getContext();  
 var collectionManager = context.getCollection();   
 var collectionLink = collectionManager.getSelfLink()  
   
 // create a new document.  
 collectionManager.createDocument(collectionLink,  
 {id: name, author: author},  
 function(err, documentCreated) {  
 if(err) throw new Error(err.message);  
   
 // filter documents by author  
 var filterQuery = "SELECT \* from root r WHERE r.author = 'George R.'";  
 collectionManager.queryDocuments(collectionLink,  
 filterQuery,  
 function(err, matchingDocuments) {  
 if(err) throw new Error(err.message);  
   
 context.getResponse().setBody(matchingDocuments.length);  
   
 // Replace the author name for all documents that satisfied the query.  
 for (var i = 0; i < matchingDocuments.length; i++) {  
 matchingDocuments[i].author = "George R. R. Martin";  
 // we don’t need to execute a callback because they are in parallel  
 collectionManager.replaceDocument(matchingDocuments[i].\_self,  
 matchingDocuments[i]);   
 }  
 })  
 })  
};

客户端可以将以上 JavaScript 逻辑“运送”到用于通过 HTTP POST 进行的事务性执行的数据库。有关使用 HTTP 方法的详细信息，请参阅[RESTful interactions with DocumentDB resources（与 DocumentDB 资源进行 RESTful 交互）](https://msdn.microsoft.com/library/azure/mt622086.aspx)。

client.createStoredProcedureAsync(collection.\_self, {id: "CRUDProc", body: businessLogic})  
 .then(function(createdStoredProcedure) {  
 return client.executeStoredProcedureAsync(createdStoredProcedure.resource.\_self,  
 "NoSQL Distilled",  
 "Martin Fowler");  
 })  
 .then(function(result) {  
 console.log(result);  
 },  
 function(error) {  
 console.log(error);  
 });

请注意，由于数据库本身能够识别 JSON 和 JavaScript，因此没有任何类型系统不匹配，也不需要“OR 映射”或代码生成方法。

存储过程和触发器与集合和集合中的文档通过一个明确定义的对象模型进行交互，该模型可公开当前集合的上下文。

DocumentDB 中的集合可以使用 [Azure DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 或任一[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx) 轻松创建、删除、读取或枚举数据库。DocumentDB 始终确保为读取或查询集合的元数据提供高度一致性。自动删除数据库，确保你不能访问任何文档、附件、存储过程、触发器和其中包含的 UDF。

## 存储过程、触发器和用户自定义函数 (UDF)

如前一节中所述，你可以编写应用程序逻辑以直接在数据库引擎内部的某个事务中运行。应用程序逻辑可以完全用 JavaScript 编写，并且可以作为存储过程、触发器或 UDF 来建模。存储过程或触发器内的 JavaScript 代码可以插入、替换、删除、读取或查询集合中的文档。但是，UDF 中的 JavaScript 却无法插入、替换或删除文档。UDF 可以枚举查询的结果集的文档，并生成另一个结果集。对于多租户，DocumentDB 将强制实施严格的基于保留项的资源监管。每个存储过程、触发器或 UDF 都可以获取固定量的操作系统资源来完成其工作。此外，存储过程、触发器或 UDF 不能针对外部 JavaScript 库进行链接，并且如果它们超出了分配给它们的资源预算，则将被列入黑名单。你可以通过使用 REST API 为集合注册存储过程、触发器或 UDF，也可以取消注册。注册时将预编译存储过程、触发器或 UDF，并将其存储为字节代码，以供以后执行。下一节说明了如何使用 DocumentDB JavaScript SDK 注册、执行和取消注册存储过程、触发器和 UDF。JavaScript SDK 相比于 [DocumentDB REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 是一个更简单的包装器。

### 注册存储过程

注册存储过程将通过 HTTP POST 在集合上创建新的存储过程资源。

var storedProc = {  
 id: "validateAndCreate",  
 body: function (documentToCreate) {  
 documentToCreate.id = documentToCreate.id.toUpperCase();  
   
 var collectionManager = getContext().getCollection();  
 collectionManager.createDocument(collectionManager.getSelfLink(),  
 documentToCreate,  
 function(err, documentCreated) {  
 if(err) throw new Error('Error while creating document: ' + err.message;  
 getContext().getResponse().setBody('success - created ' +   
 documentCreated.name);  
 });  
 }  
};  
  
client.createStoredProcedureAsync(collection.\_self, storedProc)  
 .then(function (createdStoredProcedure) {  
 console.log("Successfully created stored procedure");  
 }, function(error) {  
 console.log("Error");  
 });

### 执行存储过程

执行存储过程是针对现有的存储过程资源通过将参数传递给请求正文中的过程发出 HTTP POST 而实现的。

var inputDocument = {id : "document1", author: "G. G. Marquez"};  
client.executeStoredProcedureAsync(createdStoredProcedure.resource.\_self, inputDocument)  
 .then(function(executionResult) {  
 assert.equal(executionResult, "success - created DOCUMENT1");  
 }, function(error) {  
 console.log("Error");  
 });

### 取消注册存储过程

取消注册存储过程只需通过针对现有的存储过程资源发出 HTTP DELETE。

client.deleteStoredProcedureAsync(createdStoredProcedure.resource.\_self)  
 .then(function (response) {  
 return;  
 }, function(error) {  
 console.log("Error");  
 });

### 注册预触发器

注册触发器将通过 HTTP POST 在集合上创建新的触发器资源。你可以指定触发器是前触发还是后触发，也可以指定与之关联的操作类型（例如创建、替换、删除或全部）。

var preTrigger = {  
 id: "upperCaseId",  
 body: function() {  
 var item = getContext().getRequest().getBody();  
 item.id = item.id.toUpperCase();  
 getContext().getRequest().setBody(item);  
 },  
 triggerType: TriggerType.Pre,  
 triggerOperation: TriggerOperation.All  
}  
  
client.createTriggerAsync(collection.\_self, preTrigger)  
 .then(function (createdPreTrigger) {  
 console.log("Successfully created trigger");  
 }, function(error) {  
 console.log("Error");  
 });

### 执行前触发

触发器的执行是通过在通过请求标头发出文档资源的 POST/PUT/DELETE 请求时指定现有触发器名称完成的。

client.createDocumentAsync(collection.\_self, { id: "doc1", key: "Love in the Time of Cholera" }, { preTriggerInclude: "upperCaseId" })  
 .then(function(createdDocument) {  
 assert.equal(createdDocument.resource.id, "DOC1");  
 }, function(error) {  
 console.log("Error");  
 });

### 取消注册前触发

取消注册触发器只需通过针对现有的触发器资源发出 HTTP DELETE。

client.deleteTriggerAsync(createdPreTrigger.\_self);  
 .then(function(response) {  
 return;  
 }, function(error) {  
 console.log("Error");  
 });

### 注册 UDF

注册 UDF 将通过 HTTP POST 在集合上创建新的UDF 资源。

var udf = {   
 id: "mathSqrt",  
 body: function(number) {  
 return Math.sqrt(number);  
 },  
};  
client.createUserDefinedFunctionAsync(collection.\_self, udf)  
 .then(function (createdUdf) {  
 console.log("Successfully created stored procedure");  
 }, function(error) {  
 console.log("Error");  
 });

### 执行作为查询的一部分的 UDF

可以指定 UDF 为 SQL 查询的一部分，将其用作一种扩展核心 [DocumentDB 的 SQL 查询语言](https://msdn.microsoft.com/library/azure/dn782250.aspx)的方法。

var filterQuery = "SELECT udf.mathSqrt(r.Age) AS sqrtAge FROM root r WHERE r.FirstName='John'";  
client.queryDocuments(collection.\_self, filterQuery).toArrayAsync();  
 .then(function(queryResponse) {  
 var queryResponseDocuments = queryResponse.feed;  
 }, function(error) {  
 console.log("Error");  
 });

### 取消注册 UDF

取消注册 UDF 只需通过针对现有的 UDF 资源发出 HTTP DELETE。

client.deleteUserDefinedFunctionAsync(createdUdf.\_self)  
 .then(function(response) {  
 return;  
 }, function(error) {  
 console.log("Error");  
 });

尽管上面的代码段演示了通过 [DocumentDB JavaScript SDK](https://github.com/Azure/azure-documentdb-js) 注册 (POST)，取消注册 (PUT)、读取/列表 (GET) 和执行 (POST)，你也可以使用 [REST API](https://msdn.microsoft.com/library/azure/dn781481.aspx) 或其他[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx)。

## 文档

你可以插入、替换、删除、读取、枚举和查询集合中的任意 JSON 文档。DocumentDB 不强制要求任何架构，并且对集合中的文档进行查询也不需要辅助索引的支持。

作为一种真正的开放式数据库服务，DocumentDB 不创建任何专用的数据类型（例如日期时间）或用于 JSON 文档的特定编码。请注意，DocumentDB 不需要任何特殊的 JSON 约定来对各种文档之间的关系进行编码；DocumentDB 的 SQL 语法提供了非常强大的分层和关系查询运算符以查询和投影文档，而无需任何特殊的注释，也不需要使用可分辨属性对文档间的关系进行编码。

至于所有其他资源，可以创建、替换、删除、读取、枚举文档，也可以轻松地使用 REST API 或任一[客户端 SDK](https://msdn.microsoft.com/library/azure/dn781482.aspx) 查询文档。删除文档将立即释放与所有嵌套附件相对应的配额。文档的读取一致性级别遵守数据库帐户中的一致性策略。可以根据你的应用程序的数据一致性要求在每个请求中重写此策略。查询文档时，读取一致性遵循集合上的索引编制模式设置。对于“一致性”，将遵循帐户的一致性策略。

## 附件和媒体

[AZURE.NOTE] 附件和媒体资源是预览功能。

DocumentDB 可通过 DocumentDB 存储二进制 blob/媒体，或将其存储到远程媒体存储区。对于被称为附件的特殊文档而言，它还可以用于表示媒体的元数据。DocumentDB 中的附件是引用存储在其他位置的媒体/blob 的特殊 (JSON) 文档。附件只是捕获存储在远程媒体存储中的媒体的元数据（例如位置、作者等）的特殊文档。

考虑到一款社交阅读应用程序，它使用 DocumentDB 来存储墨迹注释，以及与某位给定用户的电子书相关联的元数据（包含评论、重点、书签、评分、喜欢/厌恶，等等）。

* 此书本身的内容会存储媒体存储中，可作为 DocumentDB 数据库帐户的一部分，也可通过远程媒体存储区获得。
* 应用程序可能会将每个用户的元数据存储为不同的文档 - 例如 book1 的 Joe 的元数据存储在由 /colls/joe/docs/book1 引用的文档中。
* 指向用户的特定书的内容页面的附件存储在对应的文档中，如 /colls/joe/docs/book1/chapter1、/colls/joe/docs/book1/chapter2，等等。

请注意，示例使用易于理解的 ID 来表示资源层次结构。通过唯一的资源 ID 和 REST API 访问资源。

对于由 DocumentDB 管理的媒体，附件的 \_media 属性将通过其 URI 引用媒体。DocumentDB 将确保在删除所有未完成的引用时对媒体进行垃圾收集。当你上传新的媒体并填充 \_media 以指向新添加的媒体时，DocumentDB 自动生成附件。如果你选择将媒体存储在由你（例如 OneDrive、Azure 存储空间、DropBox 等）管理的远程 blob 存储区，你仍可以使用附件以引用媒体。在这种情况下，你将自行创建附件并填充其 \_media 属性。

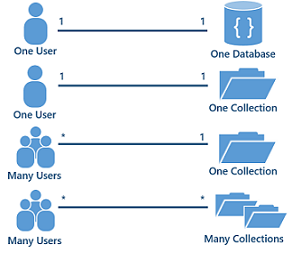
至于所有其他资源，可以创建、替换、删除、读取、枚举附件，也可以轻松地使用 REST API 或任一客户端 SDK 查询附件。至于文档，附件的读取一致性级别遵守数据库帐户中的一致性策略。可以根据你的应用程序的数据一致性要求在每个请求中重写此策略。查询附件时，读取一致性遵循集合上的索引编制模式设置。对于“一致性”，将遵循帐户的一致性策略。 ## 用户 DocumentDB 用户是指对权限进行分组的逻辑命名空间。DocumentDB 用户可能与标识管理系统或预定义应用程序角色中的用户相对应。对于 DocumentDB，用户只是表示在数据库下对一组权限进行分组的抽象。

为了在你的应用程序中实现多租户，可以在对应实际用户或你的应用程序租户的 DocumentDB 中创建用户。然后，你可以为对应针对多个集合、文档、附件等内容的访问控制的给定用户创建权限。

由于应用程序需要随用户增加而进行扩展，因此可以采用多种方法对数据进行分片。可以按照如下方法对每个用户进行建模：

* 将每个用户映射到数据库。
* 将每个用户映射到集合。
* 将对应于多个用户的文档转到专用的集合。
* 将对应于多个用户的文档转到一组集合。

无论你选择的特定分片策略是什么，你都可以将实际用户建模为 DocumentDB 数据库中的用户并将细化的权限和每个用户关联起来。

  
**分片策略和用户建模**

和所有其他资源一样，可以使用 REST API 或任一客户端 SDK 轻松创建、替换、删除、读取或枚举 DocumentDB 中的用户。DocumentDB 始终保证对读取或查询用户资源的元数据提供高度一致性。需要指出的是，自动删除用户可确保你不能访问其中所包含的任何权限。即使 DocumentDB 将权限配额声明为后台中已删除用户的一部分，你也仍然可以使用已删除的权限。

## 权限

由于数据库帐户、数据库、用户和权限等资源需要管理权限，因此从访问控制方面而言，它们都被视为管理资源。另一方面，集合、文档、附件、存储过程、触发器和 UDF 等资源都只局限在某个给定的数据库内，因此被视为应用程序资源。授权模型定义了两种类型的访问密钥：主密钥和资源密钥以对应两种类型的资源和访问它们（即管理员和用户）的角色。主密钥是数据库帐户的一部分，并且提供给设置数据库帐户的开发人员（或管理员）。此主密钥具有管理员语义，因为它可用于向管理和应用程序资源授予访问权限。与此相反，资源密钥是允许访问特定应用程序资源的精细访问密钥。因此，它会为特定资源（例如集合、文档、附件、存储过程、触发器或 UDF）捕获数据库的用户和用户权限之间的关系。

获取资源密钥的唯一方法是通过创建给定用户下的权限资源。请注意，若要创建或检索某个权限，主密钥必须呈现在授权标头中。权限资源与资源及其访问权限和用户密切相关。创建权限资源后，用户只需要提供关联的资源密钥就能获得相关资源的访问权限。因此，可以将资源密钥看作权限资源的逻辑和紧凑表示形式。

就像所有其他资源一样，可以使用 REST API 或任一客户端 SDK 轻松创建、替换、删除、读取或枚举 DocumentDB 中的权限。DocumentDB 始终保证对读取或查询权限的元数据提供高度一致性。

## 后续步骤

有关通过 HTTP 命令使用资源的详细信息，请参阅 [RESTful interactions with DocumentDB resources（RESTful 与 DocumentDB 资源的交互）](https://msdn.microsoft.com/library/azure/mt622086.aspx)。