# NoSQL Node.js 教程：DocumentDB Node.js 控制台应用程序

[AZURE.SELECTOR] - [.NET](/documentation/articles/documentdb-get-started) - [Node.js](/documentation/articles/documentdb-nodejs-get-started)

欢迎使用 DocumentDB Node.js SDK 的 Node.js 教程！ 学习本教程后，你将拥有一个创建并查询 DocumentDB 资源（包括节点数据库）的控制台应用程序。

我们将介绍：

* 创建并连接到 DocumentDB 帐户
* 设置应用程序
* 创建节点数据库
* 创建集合
* 创建 JSON 文档
* 查询集合
* 删除节点数据库

没有时间？ 不必担心！ 可在 [GitHub](https://github.com/Azure-Samples/documentdb-node-getting-started) 上获取完整的解决方案。有关快速说明，请参阅[获取完整解决方案](#GetSolution)。

完成 Node.js 教程后，请使用位于本页顶部和底部的投票按钮向我们提供反馈。如果你希望我们直接与你联系，欢迎将你的电子邮件地址附在评论中。

现在，让我们开始吧！

## Node.js 教程的先决条件

请确保你具有以下内容：

* 有效的 Azure 帐户。如果你没有，可以注册 [Azure 试用版](/pricing/free-trial/)。
* [Node.js](https://nodejs.org/) 版本 v0.10.29 或更高版本。

## 第 1 步：创建 DocumentDB 帐户

让我们创建一个 DocumentDB 帐户。如果已经有想要使用的帐户，可以跳到[安装 Node.js 应用程序](#SetupNode)。

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

## 步骤 2：安装 Node.js 应用程序

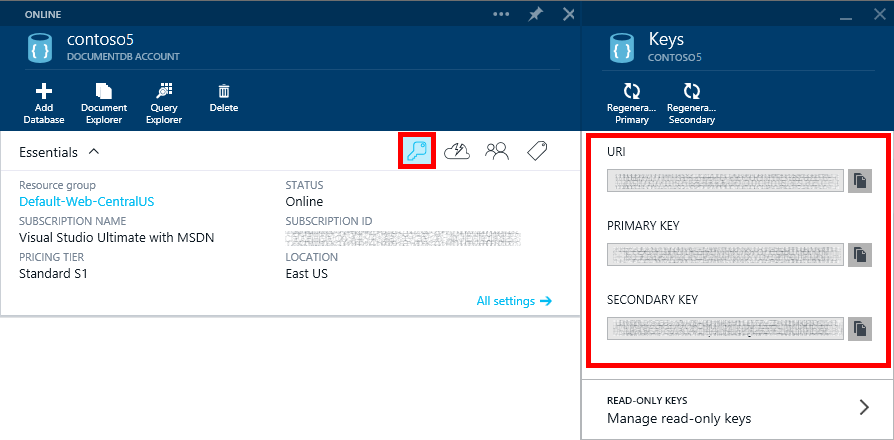
1. 打开你最爱的终端。
2. 找到你想要在其中保存 Node.js 应用程序的文件夹或目录。
3. 使用以下命令创建两个空的 JavaScript 文件：
   * Windows:
     + fsutil file createnew app.js 0
     + fsutil file createnew config.js 0
   * Linux/OS X：
     + touch app.js
     + touch config.js
4. 通过 npm 安装 documentdb 模块。请使用以下命令：
   * npm install documentdb --save

很好！ 既然已完成安装设置，让我们开始编写一些代码。

## 步骤 3：设置应用程序的配置

在喜爱的文本编辑器中打开 config.js

然后，创建名为 config 的空对象并将属性 config.endpoint 和 config.authKey 设置为 DocumentDB 终结点和授权密钥。两种配置都可以在 [Azure 门户](https://portal.azure.cn)中找到。



Node.js 教程 - Azure 门户的屏幕截图，显示 DocumentDB 帐户，在“DocumentDB 帐户”边栏选项卡上突出显示“ACTIVE”中心、“密钥”按钮，在“密钥”边栏选项卡上突出显示 URI、主密钥、辅助密钥的值 - 节点数据库

var config = {}  
  
config.endpoint = "https://YOUR\_ENDPOINT\_URI.documents.azure.com:443/";  
config.authKey = "oqTveZeWlbtnJQ2yMj23HOAViIr0ya\*\*\*\*YOUR\_AUTHORIZATION\_KEY\*\*\*\*ysadfbUV+wUdxwDAZlCfcVzWp0PQg==";

现在让我们将 database id、collection id 和 JSON documents 添加到你的 config 对象。在设置 config.endpoint 和 config.authKey 属性的下方，添加以下代码。如果已经有想要在数据库中存储的数据，则可以使用 DocumentDB 的[数据迁移工具](documentdb-import-data.md)而不是添加文档定义。

config.database = {  
 "id": "FamilyDB"  
};  
  
config.collection = {  
 "id": "FamilyColl",  
 "partitionKey": { "paths": ["/district"], "kind": "Hash" }  
};  
  
config.documents = {  
 "Andersen": {  
 "id": "Anderson.1",  
 "lastName": "Andersen",  
 "district": "WA5",  
 "parents": [{  
 "firstName": "Thomas"  
 }, {  
 "firstName": "Mary Kay"  
 }],  
 "children": [{  
 "firstName": "Henriette Thaulow",  
 "gender": "female",  
 "grade": 5,  
 "pets": [{  
 "givenName": "Fluffy"  
 }]  
 }],  
 "address": {  
 "state": "WA",  
 "county": "King",  
 "city": "Seattle"  
 }  
 },  
 "Wakefield": {  
 "id": "Wakefield.7",  
 "district": "NY23",  
 "parents": [{  
 "familyName": "Wakefield",  
 "firstName": "Robin"  
 }, {  
 "familyName": "Miller",  
 "firstName": "Ben"  
 }],  
 "children": [{  
 "familyName": "Merriam",  
 "firstName": "Jesse",  
 "gender": "female",  
 "grade": 8,  
 "pets": [{  
 "givenName": "Goofy"  
 }, {  
 "givenName": "Shadow"  
 }]  
 }, {  
 "familyName": "Miller",  
 "firstName": "Lisa",  
 "gender": "female",  
 "grade": 1  
 }],  
 "address": {  
 "state": "NY",  
 "county": "Manhattan",  
 "city": "NY"  
 },  
 "isRegistered": false  
 }  
};

数据库、集合和文档定义将充当 DocumentDB database id、collection id 和文档的数据。

最后，导出你的 config 对象，以便你可以在 app.js 文件中引用。

module.exports = config;

## 步骤 4：连接到 DocumentDB 帐户

在文本编辑器中打开空的 app.js 文件。导入 documentdb 模块，以及新创建的 config 模块。

"use strict";  
  
var documentClient = require("documentdb").DocumentClient;  
var config = require("./config");  
var url = require('url');

接下来，我们将使用之前保存的 config.endpoint 和 config.authKey 以创建新的 DocumentClient。

var client = new documentClient(config.endpoint, { "masterKey": config.authKey });

已经连接到 DocumentDB 帐户了，就让我们看看使用 DocumentDB 资源的情况。

## 步骤 5：创建节点数据库

可以通过使用 **DocumentClient** 类的 [createDatabase](https://azure.github.io/azure-documentdb-node/DocumentClient.html) 函数创建[数据库](/documentation/articles/documentdb-resources#databases)。数据库是跨集合分区的文档存储的逻辑容器。添加函数以在 app.js 文件中创建新数据库，其中 id 在 config 对象中指定。我们将首先检查以确保不存在具有相同 FamilyRegistry ID 的数据库。如果确实存在，我们将返回该数据库而不是创建新的。

var HttpStatusCodes = { NOTFOUND: 404 };  
var databaseUrl = `dbs/${config.database.id}`;  
var collectionUrl = `${databaseUrl}/colls/${config.collection.id}`;  
  
/\*\*  
 \* Get the database by ID, or create if it doesn't exist.  
 \* @param {string} database - The database to get or create  
 \*/  
function getDatabase() {  
 console.log(`Getting database:\n${config.database.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.readDatabase(databaseUrl, (err, result) => {  
 if (err) {  
 if (err.code == HttpStatusCodes.NOTFOUND) {  
 client.createDatabase(config.database, (err, created) => {  
 if (err) reject(err)  
 else resolve(created);  
 });  
 } else {  
 reject(err);  
 }  
 } else {  
 resolve(result);  
 }  
 });  
 });  
}

## 步骤 6：创建集合

[AZURE.WARNING] **CreateDocumentCollectionAsync** 将创建新的集合，它牵涉定价。有关详细信息，请访问我们的[定价页](/pricing/details/documentdb/)。

可以通过使用 **DocumentClient** 类的 [createCollection](https://azure.github.io/azure-documentdb-node/DocumentClient.html) 函数创建[集合](/documentation/articles/documentdb-resources#collections)。集合是 JSON 文档和相关联的 JavaScript 应用程序逻辑的容器。添加函数以在 app.js 文件中创建新集合，其中 id 在 config 对象中指定。同样，我们将首先检查以确保不存在具有相同 FamilyCollection ID 的集合。如果确实存在，我们将返回该集合而不是创建新的。

/\*\*  
 \* Get the collection by ID, or create if it doesn't exist.  
 \*/  
function getCollection() {  
 console.log(`Getting collection:\n${collection.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.readCollection(collectionUrl, (err, result) => {  
 if (err) {  
 if (err.code == HttpStatusCodes.NOTFOUND) {  
 client.createCollection(databaseUrl, config.collection, { offerThroughput: 400 }, (err, created) => {  
 if (err) reject(err)  
 else resolve(created);  
 });  
 } else {  
 reject(err);  
 }  
 } else {  
 resolve(result);  
 }  
 });  
 });  
}

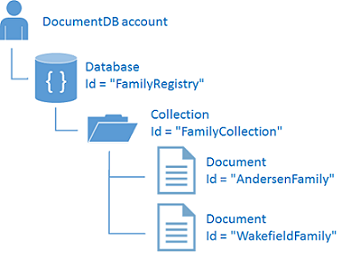
## 步骤 7：创建文档

可以通过使用 **DocumentClient** 类的 [createDocument](https://azure.github.io/azure-documentdb-node/DocumentClient.html) 函数创建[文档](/documentation/articles/documentdb-resources#documents)。文档为用户定义的（任意）JSON 内容。现在，你可以将文档插入 DocumentDB。

接下来，添加函数到 app.js，用于创建包含保存在 config 对象中的 JSON 数据的文档。同样，我们将首先检查以确保不存在具有相同 ID 的文档。

/\*\*  
 \* Get the document by ID, or create if it doesn't exist.  
 \* @param {function} callback - The callback function on completion  
 \*/  
function getFamilyDocument(document) {  
 let documentUrl = `${collectionUrl}/docs/${document.id}`;  
 console.log(`Getting document:\n${document.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.readDocument(documentUrl, { partitionKey: document.district }, (err, result) => {  
 if (err) {  
 if (err.code == HttpStatusCodes.NOTFOUND) {  
 client.createDocument(collectionUrl, document, (err, created) => {  
 if (err) reject(err)  
 else resolve(created);  
 });  
 } else {  
 reject(err);  
 }  
 } else {  
 resolve(result);  
 }  
 });  
 });  
};

祝贺你！ 你现在有函数来创建 DocumentDB 中的数据库、集合和文档！



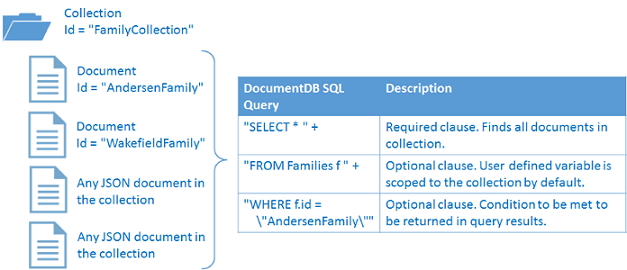
Node.js 教程 - 说明帐户、数据库、集合和文档间层次关系的关系图 - 节点数据库

## 步骤 8：查询 DocumentDB 资源

DocumentDB 对存储在每个集合中的 JSON 文档都支持[丰富查询](documentdb-sql-query.md)。下面的示例代码演示你可针对集合中文档运行的查询。向 app.js 文件添加以下函数：DocumentDB 支持类似 SQL 的查询，如下所示。有关构建复杂查询的详细信息，请参阅 [Query Playground（查询板块）](https://www.documentdb.com/sql/demo)和[查询文档](/documentation/articles/documentdb-sql-query)。

/\*\*  
 \* Query the collection using SQL  
 \*/  
function queryCollection() {  
 console.log(`Querying collection through index:\n${config.collection.id}\n`);  
  
 return new Promise((resolve, reject) => {  
 client.queryDocuments(  
 collectionUrl,  
 'SELECT VALUE r.address.city FROM root r WHERE r.lastName = "Andersen"',  
 { enableCrossPartitionQuery: true }  
 ).toArray((err, results) => {  
 if (err) reject(err)  
 else {  
 for (var queryResult of results) {  
 console.log(`Query returned ${queryResult}`);  
 }  
 resolve(results);  
 }  
 });  
 });  
};

下图说明了如何对你创建的集合调用 DocumentDB SQL 查询语法。



Node.js 教程 - 说明查询的范围和含义的关系图 - 节点数据库

[FROM](/documentation/articles/documentdb-sql-query/#from-clause) 关键字在查询中是可选的，因为 DocumentDB 查询的范围已限制为单个集合。因此，“FROM Families f”可与“FROM root r”或者任何其他所选变量名进行交换。默认情况下，DocumentDB 将推断你选择的 Families、root 或变量名，并默认引用当前集合。

## 步骤 9：删除节点数据库

删除已创建的数据库将删除该数据库及其所有子资源（集合、文档等）。通过添加下面的代码片段可以删除数据库。

/\*\*  
 \* Cleanup the database and collection on completion  
 \*/  
function cleanup() {  
 console.log(`Cleaning up by deleting database ${config.database.id}`);  
  
 return new Promise((resolve, reject) => {  
 client.deleteDatabase(databaseUrl, (err) => {  
 if (err) reject(err)  
 else resolve(null);  
 });  
 });  
}

## 步骤 10：放在一起

既然已经为应用程序设置了所有必要的功能，让我们开始调用它们吧！

将以下代码片段添加到 app.js 中的代码底部。

/\*\*  
 \* Exit the app with a prompt  
 \* @param {message} message - The message to display  
 \*/  
function exit(message) {  
 console.log(message);  
 console.log('Press any key to exit');  
 process.stdin.setRawMode(true);  
 process.stdin.resume();  
 process.stdin.on('data', process.exit.bind(process, 0));  
}  
  
getDatabase()  
 .then(() => getCollection())  
 .then(() => getFamilyDocument(config.documents.Andersen))  
 .then(() => getFamilyDocument(config.documents.Wakefield))  
 .then(() => queryCollection())  
 .then(() => cleanup())  
 .then(() => { exit(`Completed successfully`); })  
 .catch((error) => { exit(`Completed with error ${JSON.stringify(error)}`) });

## 步骤 11：运行 Node.js 应用程序！

现在，你已准备就绪，可以运行 Node.js 应用程序了。

在你的终端中，找到你的 app.js 文件并运行命令：node app.js

你应该看到已启动应用的输出。输出应该匹配下面的示例文本。

Getting database:  
FamilyDB  
  
Getting collection:  
FamilyColl  
  
Getting document:  
Anderson.1  
  
Getting document:  
Wakefield.7  
  
Querying collection through index:  
FamilyColl  
 Query returned Seattle  
  
Cleaning up by deleting database FamilyDB  
  
Completed successfully  
Press any key to exit

祝贺你！ 你已完成 Node.js 教程并已成功创建，拥有了第一个 DocumentDB 控制台应用程序！

## 获取完整的 Node.js 教程解决方案

若要生成包含本文所有示例的 GetStarted 解决方案，你将需要以下内容：

* [DocumentDB 帐户](documentdb-create-account.md)。
* GitHub 上可用的 [GetStarted](https://github.com/Azure-Samples/documentdb-node-getting-started) 解决方案。

通过 npm 安装 **documentdb** 模块。请使用以下命令： \* npm install documentdb --save

如[步骤 3：设置应用的配置](#Config)中所述，接下来在 config.js 文件中更新 config.endpoint 和 config.authKey 的值。

## 后续步骤

* 想要更复杂的 Node.js 示例？ 请参见[使用 DocumentDB 生成 Node.js Web 应用程序](/documentation/articles/documentdb-nodejs-application)。
* 了解如何[监视 DocumentDB 帐户](/documentation/articles/documentdb-monitor-accounts)。
* 在 [Query Playground（查询板块）](https://www.documentdb.com/sql/demo)中对示例数据集运行查询。
* 了解有关编程模型的详细信息，请参阅 [DocumentDB 文档页面](/services/documentdb/)的开发部分。