ALIBABA CLOUD

阿里云

小程序云

开发指南

文档版本：20220402

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SDK接入文档 07
1.1. 变更历史 07
1.2. 安装Node.js客户端SDK 07
1.3. 安装客户端SDK3.1版本 08
1.4. 安装客户端SDK3.0版本 15
1.5. 安装客户端SDK2.3版本 21
1.6. 安装客户端SDK2.2版本 23
1.7. 安装客户端SDK2.1版本 26
1.8. 安装客户端SDK2.0版本 27
1.9. 安装客户端SDK1.0版本 29
2.客户端API文档（3.x） 30
2.1. 云函数 30
2.1.1. invoke 30
2.2. 数据存储 31
2.2.1. aggregate 31
2.2.2. count 33
2.2.3. deleteOne 34
2.2.4. deleteMany 35
2.2.5. distinct 35
2.2.6. find 36
2.2.7. findOne 38
2.2.8. findOneAndDelete 39
2.2.9. findOneAndReplace 40
2.2.10. findOneAndUpdate 42
2.2.11. replaceOne 44
2.2.12. insertMany 45

2.2.13. insertOne 45
2.2.14. updateMany 46
2.2.15. updateOne 47
2.3. 文件存储 48
2.3.1. uploadFile 48
2.3.2. deleteFile 51
2.4. 用户管理 52
2.4.1. getInfo 52
3.客户端API文档（2.x） 54
3.1. 云函数 54
3.1.1. invoke 54
3.2. 数据存储 54
3.2.1. aggregate 54
3.2.2. count 56
3.2.3. deleteOne 57
3.2.4. deleteMany 57
3.2.5. distinct 58
3.2.6. find 58
3.2.7. findOne 59
3.2.8. findOneAndDelete 60
3.2.9. findOneAndReplace 61
3.2.10. findOneAndUpdate 62
3.2.11. replaceOne 63
3.2.12. insertMany 64
3.2.13. insertOne 64
3.2.14. updateMany 65
3.2.15. updateOne 66
3.3. 文件存储 66

3.3.1. uploadFile 67
3.3.2. deleteFile 68
3.4. 用户管理 69
3.4.1. getInfo 69
3.4.2. authorize 70

1.SDK接入文档

1.1. 变更历史

本文记录了各版本SDK的新增特性。

版本	版本号	说明
客户端SDK 3.1.1	@alicloud/mpserverless-sdk@3.1.1	数据存储findOneAndUpdate操作新增支持returnNewDocument参数。更多信息， 请参见 安装客户端SDK3.1版本。
客户端SDK 3.1	@alicloud/mpserverless-sdk@3.1.0	微信小程序侧BUG修复。更多信息，请参见安装客户端SDK3.1版本。
客户端SDK 3.0	@alicloud/mpserverless-sdk@3.0.0	改动如下： • SDK架构升级。 • 支付宝、微信、钉钉依赖相同版本SDK。 • SDK创建实例参数变更。 • 去除原有的授权接口，新增SDK初始化接口。 更多信息，请参见安装客户端SDK3.0版本。
客户端SDK 2.3	@alicloud/mpserverless-sdk@2.3.2	支付宝小程序修改为静默授权。更多信息，请参见安装客户端SDK2.3版本。
客户端SDK 2.2	@alicloud/mpserverless-sdk@2.2.0	增加统计分析功能。更多信息，请参见安装客户端SDK2.2版本。
客户端SDK 2.1	@alicloud/mpserverless-sdk@2.1.0	增加匿名用户授权方式。 更多信息，请参见安装客户端SDK2.1版本。
客户端SDK 2.0	@alicloud/mpserverless-sdk@2.0.0	支持在微信小程序客户端中使用小程序云Serverless服务。 更多信息，请参见在微信小程序中使用SDK。
客户端SDK 1.0	@alicloud/mpserverless-sdk@1.0.3	通过小程序SDK可以直接在支付宝IDE中调用小程序云的云函数、云存储和数据库等服务。

1.2. 安装Node.js客户端SDK

本文介绍如何集成小程序Serverless Node.js SDK，集成后您可以在Node.js工程中访问小程序Serverless服务。

前提条件

在首次使用小程序云服务前，您需要开通小程序云服务。详细信息，请参见 开通小程序云服务。

操作步骤

1. 安装SDK，在Node.js项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-node-sdk@1.0.0 --save
```

2. 初始化SDK，在项目中加入如下SDK初始化代码。

```
// SDK引用 const MPServerless = require('@alicloud/mpserverless-node-sdk').default; // SDK初始化 const client = new MPServerless({ spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识 endpoint: 'https://endpoint', // 服务空间地址 serverSecret: '-----BEGIN PRIVATE KEY-----\nMIIEvgIBADANBgkqhki******G9w0BAQE7V0sQaj\n-----END PRIVATE KEY-----', // 鉴权Secret Key });
```

其中：

- spaceId 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。
- serverSecret （鉴权Secret Key）可通过调用DescribeSpaceClientConfig接口获取。

3. 调用SDK，以下为Node.js调用小程序Serverless SDK示例代码，详情请参见云函数API文档和数据存储API文档。

```
// 云函数 try { const res = await client.function.invoke('testFunction'); console.log('function invoke result: ', res); } catch (err) { console.log('function invoke error: ', err); } // 云数据库 try { const res = await client.db.collection('testTable').count({}); console.log('db service result: ', res); } catch (err) { console.log('db service error: ', err); }
```

> **说明** 异常处理说明：
> - 通过 try/catch 捕获调用错误。
> - 捕获错误error中包括 error.name 和 error.message 字段。

1.3. 安装客户端SDK3.1版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。

前提条件

- 在首次使用小程序云服务前，您需要开通小程序云服务。详情信息，请参见开通小程序云服务。
- 在使用小程序云SDK前，请确保已经正确安装了Node.js开发环境。详情信息，请参见Node.js官方安装教程。

在支付宝小程序中使用SDK

完成以下操作，在支付宝小程序中使用SDK。

1. 在支付宝小程序项目的根目录，执行以下命令安装SDK。

```
npm install --save @alicloud/mpserverless-sdk@3.1.1
```

2. 在支付宝小程序项目的配置文件mini.project.json中，添加以下配置信息。

```
// mini.project.json
"enableNodeModuleBabelTransform": true,
"node_modules_es6_whitelist": [ "@alicloud/mpserverless-sdk"]
```

3. 在小程序项目中的app.js中添加以下代码，构造小程序云SDK的实例对象。

```
// app.js
import MPServerless from '@alicloud/mpserverless-sdk'

const mpserverless = new MPServerless(my, {
 appId: <您的支付宝小程序appId>,
  spaceId: <您的小程序云serverless服务空间spaceId>
  clientSecret: <您的小程序云serverless服务空间clientSecret>
  endpoint: <您的小程序云serverless服务空间Api endpoint>
});
```

其中：

- my是支付宝小程序开发者工具中的全局基础库，可以在开发环境中直接获取。
- appId是小程序的ID。您可以在支付宝小程序开发者工具设置中，直接获取小程序的App ID。

- spaceId、clientSecret和endpoint在小程序Serverless控制台创建服务空间后可以获得。相关操作，请参见创建服务空间。

4. 在app.js的onLaunch生命周期中调用init接口完成初始化。

> **说明**
>
> 在小程序端开始使用Serverless服务前，您需先调用mpserverless.init方法完成服务的初始化。在小程序中，我们建议您在onLaunch生命周期中进行初始化操作，并将小程序云SDK的实例对象挂载到小程序的全局App对象之上，以便后续在其他文件中调用。

```
// app.js
App({
  mpserverless: mpserverless,
  onLaunch() {
    mpserverless.init();
  },
});
```

5. 在其他文件中使用SDK。

> **说明**
>
> 在其他文件中可以通过getApp的方式，从全局对象APP中获取SDK的实例对象。

```
// 其他文件中使用 sdk
const { mpserverless } = getApp();

mpserverless.db.collection('test').find();
mpserverless.function.invoke('test');
```

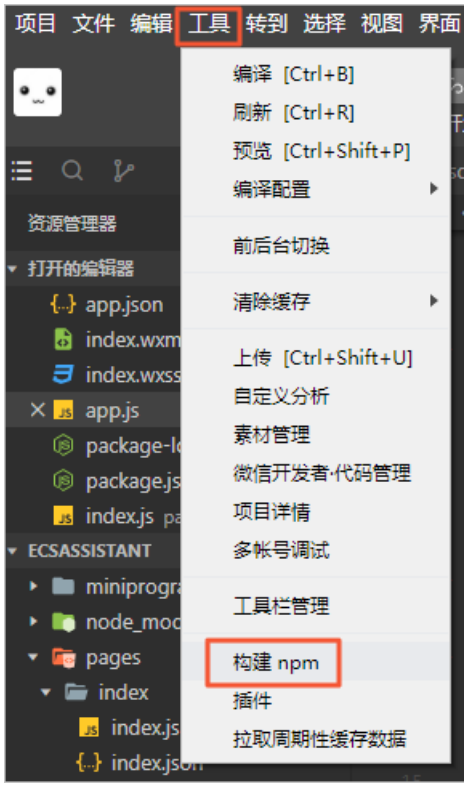
在微信小程序中使用SDK

完成以下操作，在微信小程序中使用小程序云SDK。

1. 在微信小程序项目的根目录，执行以下命令安装SDK。

```
npm install --save @alicloud/mpserverless-sdk@3.1.1
```

2. 在微信小程序IDE中，选择**工具>构建npm**。



3. 在小程序项目中的app.js中添加以下代码，构造小程序云SDK的实例对象。

```
// app.js
import MPServerless from '@alicloud/mpserverless-sdk';

const mpserverless = new MPServerless(wx, {
 appId: <您的微信小程序appId>,
  spaceId: <您的小程序云serverless服务空间spaceId>
  clientSecret: <您的小程序云serverless服务空间clientSecret>
  endpoint: <您的小程序云serverless服务空间Api endpoint>
});
```

其中：

- wx是微信小程序IDE中的全局基础库，可以在开发环境中直接获取。
- appId是小程序的ID。您可以在微信开发者工具设置中，直接获取小程序的App ID。

- spaceId、client Secret和endpoint在小程序Serverless控制台创建服务空间后可以获得。相关操作，请参见创建服务空间。

4. 在app.js的onLaunch生命周期中调用init接口完成初始化。

> **说明**
>
> 在小程序端开始使用Serverless服务前，您需先调用mpserverless.init方法完成服务的初始化。在小程序中，我们建议您在onLaunch生命周期中进行初始化操作，并将小程序云SDK的实例对象挂载到小程序的全局App对象之上，以便后续在其他文件中调用。

```
App({
  mpserverless: mpserverless,
  onLaunch(options) {
    mpserverless.init();
  },
});
```

5. 在其他文件中使用SDK。

> **说明**
>
> 在其他文件中可以通过getApp的方式，从全局对象APP中获取SDK的实例对象。

```
// 其他文件中使用 sdk
const { mpserverless } = getApp();

mpserverless.db.collection('test').find();
mpserverless.function.invoke('test');
```

在钉钉小程序中使用SDK

完成以下操作，在第三方个人应用的钉钉小程序中使用小程序云SDK。

1. 在钉钉小程序项目的根目录，执行以下命令安装SDK。

```
npm install --save @alicloud/mpserverless-sdk@3.1.1
```

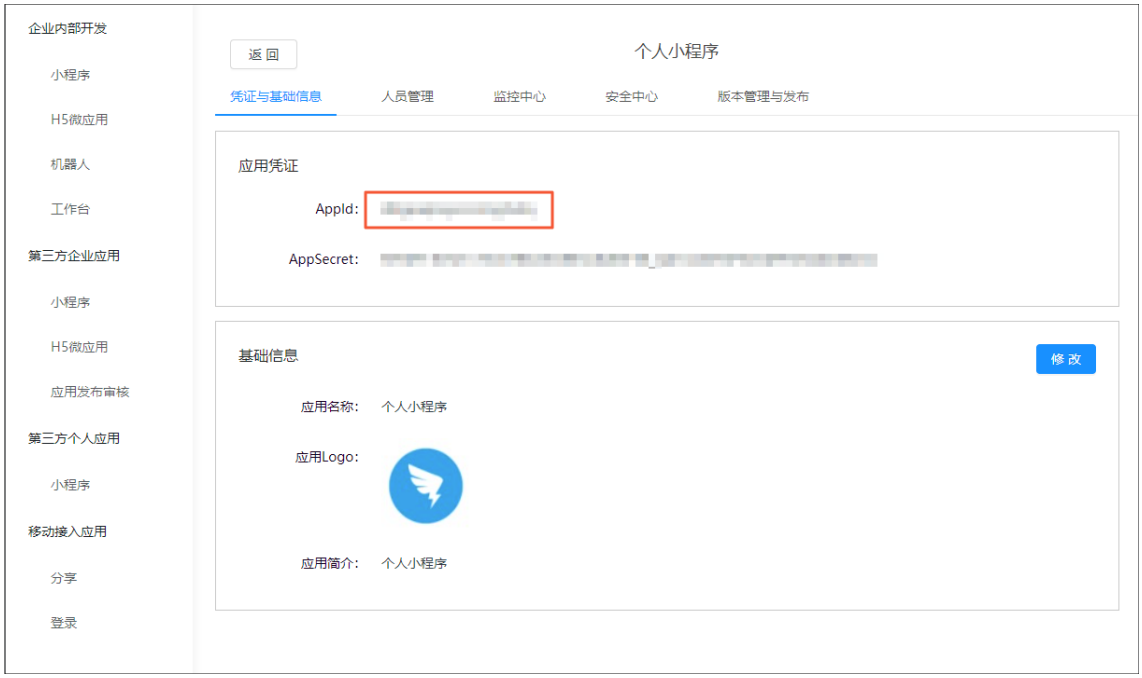
2. 在小程序项目中的app.js中添加以下代码，构造小程序云SDK的实例对象。

```
// app.js
import MPServerless from '@ali/mpserverless-sdk'

const mpserverless = new MPServerless(dd, {
 appId: <您的钉钉小程序appId>,
  spaceId: <您的小程序云serverless服务空间spaceId>
  clientSecret: <您的小程序云serverless服务空间clientSecret>
  endpoint: <您的小程序云serverless服务空间Api endpoint>
});
```

其中：

- dd是钉钉小程序开发者工具的全局基础库，可以在开发环境中直接获取。
- appId是小程序的ID。可以在钉钉开放平台获取。



- spaceId、clientSecret和endpoint在小程序Serverless控制台创建服务空间后可以获得。相关操作，请参见创建服务空间。

3. 在app.js的onLaunch生命周期中调用init接口完成初始化。

> **说明**
>
> 在小程序端开始使用Serverless服务前，您需先调用mpserverless.init方法完成服务的初始化。在小程序中，我们建议您在onLaunch生命周期中进行初始化操作，并将小程序云SDK的实例对象挂载到小程序的全局App对象之上，以便后续在其他文件中调用。

```
App({
  mpserverless: mpserverless,
  onLaunch(options) {
    await mpserverless.init();
  },
});
```

4. 在其他文件中使用SDK。

> **说明**
>
> 在其他文件中可以通过getApp的方式，从全局对象APP中获取SDK的实例对象。

```
// 其他文件中使用 sdk
const { mpserverless } = getApp();

mpserverless.db.collection('test').find();
mpserverless.function.invoke('test');
```

1.4. 安装客户端SDK3.0版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。

前提条件

- 在首次使用小程序云服务前，您需要开通小程序云服务。详情信息，请参见开通小程序云服务。
- 在使用小程序云SDK前请确保已经正确安装了Node.js开发环境。详情信息，请参见nodejs官方安装教程。

在支付宝小程序中使用SDK

完成以下操作，在支付宝小程序中使用SDK：

1. 在支付宝小程序项目的根目录执行以下命令安装SDK。

```
npm install --save @alicloud/mpserverless-sdk@3.0.0
```

2. 在小程序项目中的app.js中添加以下代码，构造小程序云SDK的实例对象。

```
// app.js
import MPServerless from '@alicloud/mpserverless-sdk'

const mpserverless = new MPServerless(my, {
    appId: '1234456789', // 小程序应用标识
    spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识
    clientSecret: '6c3c86xxxx6', // 服务空间 secret key
    endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得
});
```

其中：

- my是支付宝小程序开发者工具中的全局基础库，可以在开发环境中直接获取。
- appId是小程序的ID。您可以在支付宝小程序开发者工具中直接获取小程序的App ID。

- spaceId、clientSecret和endpoint在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 在app.js的onLaunch生命周期中调用init接口完成初始化。

> **说明**
>
> 在小程序端开始使用 serverless 服务前，需先调用mpserverless.init方法完成服务的初始化。在小程序中，我们建议您在onLaunch生命周期中进行初始化操作，并将小程序云SDK的实例对象挂载到小程序的全局APP对象之上，以便后续在其他文件中调用。

```
// app.js
App({
  mpserverless: mpserverless,
  onLaunch() {
    mpserverless.init(); // 您可以通过 result 来判断初始化完成情况
  },
});
```

4. 在其他文件中使用SDK。

> **说明**
>
> 在其他文件中可以通过getApp的方式从全局对象APP中获取SDK的实例对象。

```
// 其他文件中使用 sdk
const { mpserverless } = getApp();

mpserverless.db.collection('test').find();
mpserverless.function.invoke('test');
```

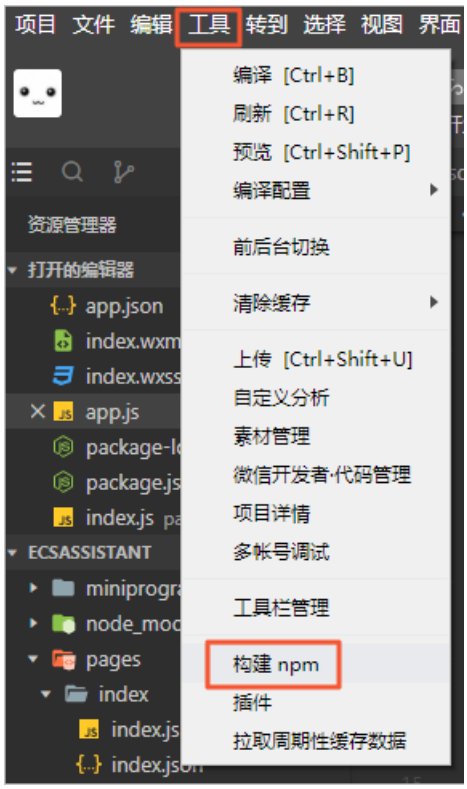
在微信小程序中使用SDK

完成以下操作，在微信小程序中使用小程序云SDK：

1. 在微信小程序项目的根目录执行以下命令安装SDK。

```
npm install --save @alicloud/mpserverless-sdk@3.0.0
```

2. 在微信小程序IDE中单击**工具>构建npm**。



3. 在小程序项目中的app.js中添加以下代码，构造小程序云SDK的实例对象。

```
// app.js
import MPServerless from '@alicloud/mpserverless-sdk'

const mpserverless = new MPServerless(wx, {
    appId: '1234456789', // 小程序应用标识
    spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识
    clientSecret: '6c3c86xxxx6', // 服务空间 secret key
    endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得
});
```

其中：

- wx是微信小程序IDE中的全局基础库，可以在开发环境中直接获取。

- appId是小程序的ID。您可以在微信开发者工具中直接获取小程序的App ID。

- spaceId、client Secret和endpoint在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

4. 在app.js的onLaunch生命周期中调用init接口完成初始化。

> **说明**
>
> 在小程序端开始使用serverless服务前，需先调用mpserverless.init方法完成服务的初始化。在小程序中，我们建议您在onLaunch生命周期中进行初始化操作，并将小程序云SDK的实例对象挂载到小程序的全局APP对象之上，以便后续在其他文件中调用。

```
App({
  mpserverless: mpserverless,
  onLaunch(options) {
    mpserverless.init(); // 您可以通过 result 来判断初始化完成情况
  },
});
```

5. 在其他文件中使用SDK。

> **说明**
>
> 在其他文件中可以通过getApp的方式从全局对象APP中获取SDK的实例对象。

```
// 其他文件中使用 sdk
const { mpserverless } = getApp();

mpserverless.db.collection('test').find();
mpserverless.function.invoke('test');
```

在钉钉小程序中使用SDK

完成以下操作，在第三方个人应用的钉钉小程序中使用小程序云SDK：

1. 在钉钉小程序项目的根目录执行以下命令安装SDK。

```
npm install --save @alicloud/mpserverless-sdk@3.0.0
```

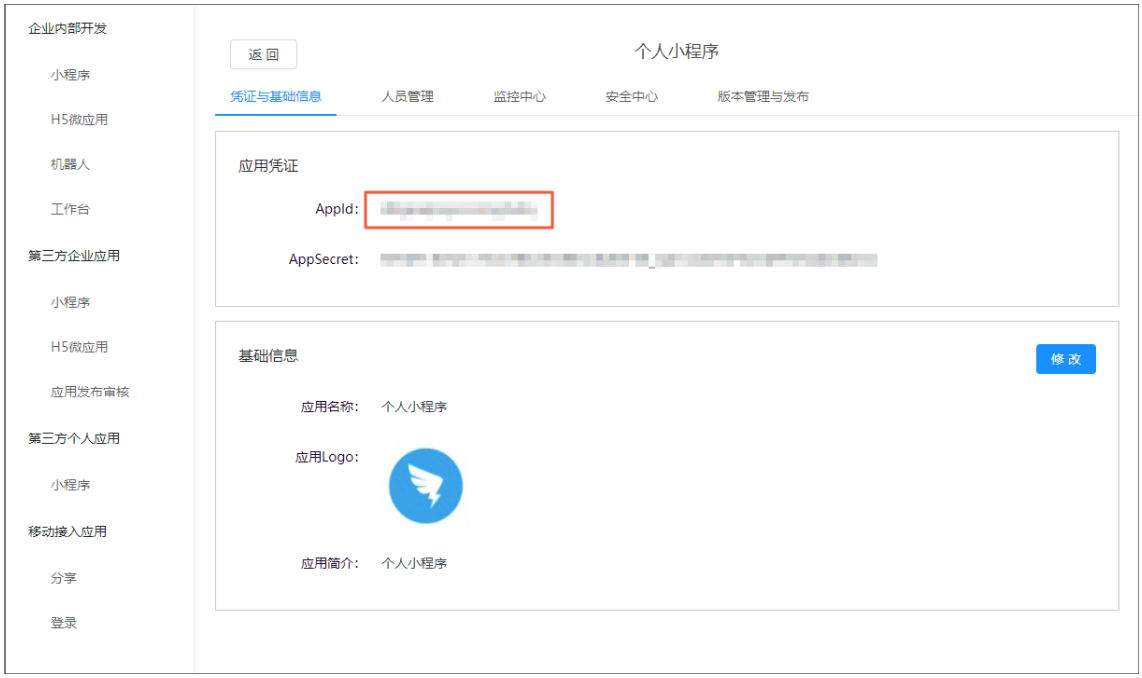
2. 在小程序项目中的app.js中添加以下代码，构造小程序云SDK的实例对象。

```
// app.js
import MPServerless from '@alicloud/mpserverless-sdk'

const mpserverless = new MPServerless(dd, {
    appId: '1234456789', // 小程序应用标识
    spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识
    clientSecret: '6c3c86xxxx6', // 服务空间 secret key
    endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得
});
```

其中：

- dd 是钉钉小程序开发者工具的全局基础库，可以在开发环境中直接获取。
- appId是小程序的ID。可以在钉钉开放平台获取。



- spaceId、clientSecret和endpoint在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 在app.js的onLaunch生命周期中调用init接口完成初始化。

> **说明**
>
> 在小程序端开始使用serverless服务前，需先调用mpserverless.init方法完成服务的初始化。在小程序中，我们建议您在onLaunch生命周期中进行初始化操作，并将小程序云SDK的实例对象挂载到小程序的全局APP对象之上，以便后续在其他文件中调用。

```
App({
  mpserverless: mpserverless,
  onLaunch(options) {
    await mpserverless.init(); // 您可以通过 result 来判断初始化完成情况
  },
});
```

4. 在其他文件中使用SDK。

> **说明**
>
> 在其他文件中可以通过getApp的方式从全局对象APP中获取SDK的实例对象。

```
// 其他文件中使用 sdk
const { mpserverless } = getApp();

mpserverless.db.collection('test').find();
mpserverless.function.invoke('test');
```

1.5. 安装客户端SDK2.3版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。2.3版本支付宝小程序修改为静默授权。

前提条件

在首次使用小程序云服务前，您需要开通小程序云服务。详细信息，请参见 开通小程序云服务。

在支付宝小程序中使用SDK

完成以下操作，在支付宝小程序中使用SDK：

1. 在支付宝小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.3.2 --save
```

2. 在小程序项目中添加以下代码，初始化SDK。

```
import MPServerless from '@alicloud/mpserverless-sdk'; const mpServerless = new MPServe
rless({ uploadFile: my.uploadFile, request: my.request, getAuthCode: my.getAuthCode, ge
tFileInfo: my.getFileInfo, getImageInfo: my.getImageInfo }, { appId: '1234456789', //
小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识 clientSecret: '6c3c8
6xxxx6', // 服务空间 secret key endpoint: 'https://endpoint', // 服务空间地址，从小程序Serv
erless控制台处获得 });
```

   其中：

   - appId 是小程序的ID。您可以在支付宝小程序开放平台获取小程序的App ID。
   - spaceId 、 clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权。

```
const res = await mpServerless.user.authorize({   authProvider: 'alipay_openapi' }); if
(res.success) {   console.log('授权成功'); }
```

其中：

> 说明
> - 如果同时指定了 authProvider 和 authType， authType 配置优先。
> - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
> - 如果没有设置 authType ，则必须设置 authProvider 。
> - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。

在微信小程序中使用SDK

完成以下操作，在微信小程序中使用小程序云SDK：

1. 在微信小程序项目的根目录执行以下命令安装SDK文件。

   npm install @alicloud/mpserverless-sdk@2.3 --save

2. 在*app.js*或其他需要的页面中引入SDK文件。

```
//app.js const MPServerless = require('/sdk/mpserverless.js'); const MPServerless = req
uire('@alicloud/mpserverless-sdk'); const mpServerless = new MPServerless({ uploadFile:
wx.uploadFile, request: wx.request, getAuthCode: wx.login, getFileInfo: wx.getFileInfo,
getImageInfo: wx.getImageInfo, }, { appId: 'appId', // 小程序应用标识 spaceId: '', // 服
务空间标识 clientSecret: '', // 服务空间 secret key endpoint: '', // 服务空间地址，从小程序
serverless 控制台处获得 });
```

   其中：

   - appId 是小程序的ID。您可以在微信公众平台获取小程序的App ID。
   - spaceId 、 clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权

```
const res = await mpServerless.user.authorize({   authProvider: 'wechat_openapi' }); if
(res.success) {   console.log('授权成功'); }
```

   其中：

> 说明
> - 如果同时指定了 authProvider 和 authType， authType 配置优先。
> - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
> - 如果没有设置 authType ，则必须设置 authProvider 。
> - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。

在钉钉小程序中使用SDK

完成以下操作，在第三方个人应用的钉钉小程序中使用小程序云SDK：

1. 在钉钉小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/dt-mpserverless-sdk@0.0.2 --save
```

2. 在小程序项目中添加以下代码，初始化SDK。

```
import MPServerlessSDK from '@alicloud/dt-mpserverless-sdk'; const mpServerless = new M
PServerlessSDK({ uploadFile: dd.uploadFile, request: dd.httpRequest, getAuthCode: dd.ge
tAuthCode }, { appId: '1234456789', // 小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxx
x', // 服务空间标识 clientSecret: '6c3c86xxxx6', // 服务空间 secret key endpoint: 'https:/
/endpoint', // 服务空间地址，从小程序Serverless控制台处获得 })
```

   其中：

   - appId 是小程序的ID。您可以在钉钉开放平台获取小程序的App ID。
   - spaceId 、 clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权。

```
const res = await mpServerless.user.authorize({   authProvider: 'dingtalk_openapi' });
if (res.success) {   console.log('授权成功'); }
```

   其中：

   > **说明**
   > - 如果同时指定了 authProvider 和 authType ， authType 配置优先。
   > - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
   > - 如果没有设置 authType ，则必须设置 authProvider 。
   > - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。

1.6. 安装客户端SDK2.2版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。2.2版本开始支持统计分析功能。

前提条件

在首次使用小程序云服务前，您需要开通小程序云服务。详细信息，请参见 开通小程序云服务。

在支付宝小程序中使用SDK

完成以下操作，在支付宝小程序中使用SDK：

1. 在支付宝小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.2 --save
```

2. 在小程序项目中添加以下代码，初始化SDK。

```
import MPServerless from '@alicloud/mpserverless-sdk'; const mpServerless = new MPServerless({ uploadFile: my.uploadFile, request: my.request, getAuthCode: my.getAuthCode, getFileInfo: my.getFileInfo, getImageInfo: my.getImageInfo }, { appId: '1234456789', // 小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识 clientSecret: '6c3c86xxxx6', // 服务空间 secret key endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得 });
```

其中：

- appId 是小程序的ID。您可以在支付宝小程序开放平台获取小程序的App ID。
- spaceId 、 clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权。

```
const res = await mpServerless.user.authorize({   authProvider: 'alipay_openapi' }); if (res.success) {   console.log('授权成功'); }
```

其中：

> 说明
>
> - 如果同时指定了 authProvider 和 authType ， authType 配置优先。
> - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
> - 如果没有设置 authType ，则必须设置 authProvider 。
> - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。
> - 用户第一次进入小程序时，授权操作会唤起**用户授权弹框**，需要用户阅读并同意《用户授权协议》，以获取用户的昵称、头像、性别、地区和用户状态等信息。如果用户拒绝了该协议，则会造成小程序Serverless三方授权失败，无法使用小程序Serverless服务。如需避免此种情况发生，请使用匿名授权方式，匿名授权相当于静默授权，不会唤起授权弹框。

在微信小程序中使用SDK

完成以下操作，在微信小程序中使用小程序云SDK：

1. 在微信小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.2 --save
```

2. 在*app.js*或其他需要的页面中引入SDK文件。

```
//app.js const MPServerless = require('/sdk/mpserverless.js'); const MPServerless = require('@alicloud/mpserverless-sdk'); const mpServerless = new MPServerless({ uploadFile: wx.uploadFile, request: wx.request, getAuthCode: wx.login, getFileInfo: wx.getFileInfo, getImageInfo: wx.getImageInfo, }, { appId: 'appId', // 小程序应用标识 spaceId: '', // 服务空间标识 clientSecret: '', // 服务空间 secret key endpoint: '', // 服务空间地址，从小程序serverless 控制台处获得 });
```

其中：

- appId 是小程序的ID。您可以在微信公众平台获取小程序的App ID。
- spaceId 、 clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参

见创建服务空间。

3. 调用user.authorize方法完成授权

```
const res = await mpServerless.user.authorize({   authProvider: 'wechat_openapi' }); if (res.success) {   console.log('授权成功'); }
```

其中：

> 说明
> - 如果同时指定了 authProvider 和 authType ， authType 配置优先。
> - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
> - 如果没有设置 authType ，则必须设置 authProvider 。
> - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。
> - 用户第一次进入小程序时，授权操作会唤起**用户授权弹框**，需要用户阅读并同意《用户授权协议》，以获取用户的昵称、头像、性别、地区和用户状态等信息。如果用户拒绝了该协议，则会造成小程序Serverless三方授权失败，无法使用小程序Serverless服务。如需避免此种情况发生，请使用匿名授权方式，匿名授权相当于静默授权，不会唤起授权弹框。

在钉钉小程序中使用SDK

完成以下操作，在第三方个人应用的钉钉小程序中使用小程序云SDK：

1. 在钉钉小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/dt-mpserverless-sdk@0.0.2 --save
```

2. 在小程序项目中添加以下代码，初始化SDK。

```
import MPServerlessSDK from '@alicloud/dt-mpserverless-sdk'; const mpServerless = new MPServerlessSDK({ uploadFile: dd.uploadFile, request: dd.httpRequest, getAuthCode: dd.getAuthCode }, { appId: '1234456789', // 小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识 clientSecret: '6c3c86xxxx6', // 服务空间 secret key endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得 })
```

其中：

- appId 是小程序的ID。您可以在钉钉开放平台获取小程序的App ID。
- spaceId 、 clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权。

```
const res = await mpServerless.user.authorize({   authProvider: 'dingtalk_openapi' }); if (res.success) {   console.log('授权成功'); }
```

其中：

说明

- 如果同时指定了 authProvider 和 authType， authType 配置优先。
- 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
- 如果没有设置 authType ，则必须设置 authProvider 。
- 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。
- 用户第一次进入小程序时，授权操作会唤起**用户授权弹框**，需要用户阅读并同意《用户授权协议》，以获取用户的昵称、头像、性别、地区和用户状态等信息。如果用户拒绝了该协议，则会造成小程序Serverless三方授权失败，无法使用小程序Serverless服务。如需避免此种情况发生，请使用匿名授权方式，匿名授权相当于静默授权，不会唤起授权弹框。

1.7. 安装客户端SDK2.1版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。自2.1版本起支持匿名用户授权方式。

前提条件

在首次使用小程序云服务前，您需要开通小程序云服务。详细信息，请参见 开通小程序云服务。

在支付宝小程序中使用SDK

完成以下操作，在支付宝小程序中使用SDK：

1. 在支付宝小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.1.0 --save
```

2. 在小程序项目中添加以下代码，初始化SDK。

```
import MPServerless from '@alicloud/mpserverless-sdk'; const mpServerless = new MPServerless({ uploadFile: my.uploadFile, request: my.request, getAuthCode: my.getAuthCode }, { appId: '1234456789', // 小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识 clientSecret: '6c3c86xxxx6', // 服务空间 secret key endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得 });
```

其中：

- appId 是小程序的ID。您可以在支付宝小程序开放平台获取小程序的App ID。
- spaceId clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权。

```
const res = await mpServerless.user.authorize({   authProvider: 'alipay_openapi' }); if (res.success) {   console.log('授权成功'); }
```

其中：

- authProvider ： 指定授权方，取值*alipay_openap*和*wechat_openapi*。
- authType ： 指定授权方式。取值*anonymous*表示通过匿名方式授权。

> 说明
>
> - 如果同时指定了 authProvider 和 authType， authType 配置优先。
> - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
> - 如果没有设置 authType ，则必须设置 authProvider 。
> - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。

在微信小程序中使用SDK

完成以下操作，在微信小程序中使用小程序云SDK：

1. 在微信小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.1 --save
```

2. 在*app.js*或其他需要的页面中引入SDK文件。

```
//app.js const MPServerless = require('/sdk/mpserverless.js'); const MPServerless = req
uire('@alicloud/mpserverless-sdk'); const mpServerless = new MPServerless({ uploadFile:
wx.uploadFile, request: wx.request, getAuthCode: wx.login, getFileInfo: wx.getFileInfo,
getImageInfo: wx.getImageInfo, }, { appId: 'appId', // 小程序应用标识 spaceId: '', // 服
务空间标识 clientSecret: '', // 服务空间 secret key endpoint: '', // 服务空间地址，从小程序
serverless 控制台处获得 });
```

   其中：

   - appId 是小程序的ID。您可以在微信公众平台获取小程序的App ID。
   - spaceId clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权

```
const res = await mpServerless.user.authorize({   authProvider: 'wechat_openapi' }); if
(res.success) {   console.log('授权成功'); }
```

   其中：

   - authProvider ：指定授权方，取值*alipay_openap*和*wechat_openapi*。
   - authType ：指定授权方式。取值*anonymous*表示通过匿名方式授权。

> 说明
>
> - 如果同时指定了 authProvider 和 authType， authType 配置优先。
> - 如果将 authType 设置为*anonymous*，则不需要设置 authProvider 。
> - 如果没有设置 authType ，则必须设置 authProvider 。
> - 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。

1.8. 安装客户端SDK2.0版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。2.0版本支持支付宝和微信小程序，在安装SDK后，您需要调用user.authorize进行授权。

前提条件

在首次使用小程序云服务前，您需要开通小程序云服务。详细信息，请参见 开通小程序云服务。

在支付宝小程序中使用SDK

完成以下操作，在支付宝小程序中使用SDK：

1. 在支付宝小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.0.0 --save
```

2. 在小程序项目中添加以下代码，初始化SDK。

```
import MPServerless from '@alicloud/mpserverless-sdk'; const mpServerless = new MPServerless({ uploadFile: my.uploadFile, request: my.request, getAuthCode: my.getAuthCode }, { appId: '1234456789', // 小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空间标识 clientSecret: '6c3c86xxxx6', // 服务空间 secret key endpoint: 'https://endpoint', // 服务空间地址，从小程序Serverless控制台处获得 });
```

   其中：

   - appId 是小程序的ID。您可以在支付宝小程序开放平台获取小程序的App ID。
   - spaceId clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权。

```
const res = await mpServerless.user.authorize({   authProvider: 'alipay_openapi' }); if (res.success) {   console.log('授权成功'); }
```

在微信小程序中使用SDK

完成以下操作，在微信小程序中使用小程序云SDK：

1. 在微信小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@2.0.0 --save
```

2. 在*app.js*或其他需要的页面中引入SDK文件。

```
//app.js const MPServerless = require('/sdk/mpserverless.js'); const MPServerless = require('@alicloud/mpserverless-sdk'); const mpServerless = new MPServerless({ uploadFile: wx.uploadFile, request: wx.request, getAuthCode: wx.login, getFileInfo: wx.getFileInfo, getImageInfo: wx.getImageInfo, }, { appId: 'appId', // 小程序应用标识 spaceId: '', // 服务空间标识 clientSecret: '', // 服务空间 secret key endpoint: '', // 服务空间地址，从小程序serverless 控制台处获得 });
```

   其中：

   - appId 是小程序的ID。您可以在微信公众平台获取小程序的App ID。
   - spaceId clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

3. 调用user.authorize方法完成授权

```
const res = await mpServerless.user.authorize({   authProvider: 'wechat_openapi' }); if
(res.success) {   console.log('授权成功'); }
```

1.9. 安装客户端SDK1.0版本

通过小程序云SDK，您可以在小程序中直接访问小程序云Serverless服务。

前提条件

在首次使用小程序云服务前，您需要开通小程序云服务。详细信息，请参见 开通小程序云服务。

操作步骤

1. 在小程序项目的根目录执行以下命令安装SDK。

```
npm install @alicloud/mpserverless-sdk@1.0.3 --save
```

2. 执行以下代码完成初始化。

```
import MPServerless from '@alicloud/mpserverless-sdk'; const mpServerless = new MPServe
rless({ uploadFile: my.uploadFile, request: my.request, getAuthCode: my.getAuthCode },
{ appId: '1234456789', // 小程序应用标识 spaceId: 'db4dd657-7041-470a-90xxxxx', // 服务空
间标识 clientSecret: '6c3c86xxxx6', // 服务空间 secret key endpoint: 'https://endpoint',
// 服务空间地址，从小程序Serverless控制台处获得 });
```

其中：

- appId 是小程序的ID。您可以在支付宝小程序控制台获得。
- spaceId clientSecret 和 endpoint 在小程序Serverless控制台创建服务空间后可以获得。详情参见创建服务空间。

2.客户端API文档（3.x）

2.1. 云函数

2.1.1. invoke

调用云函数。

方法定义

```
mpserverless.function.invoke(functionName: string, functionArgs?: object): Promise<Result>
```

请求参数

字段名	类型	必填	说明
functionName	String	是	所调用的云函数名。
functionArgs	Object	否	目标云函数的参数。

返回参数

字段名	类型	说明
success	Boolean	执行状态。
requestId	String	请求ID。
result	Any	接口返回内容，由开发者代码和请求参数 header 的 content-type 决定，默认 content-type 为 application/json。

示例

假设已经有了一个云函数sum，云函数的定义如下。

```
'use strict';

module.exports = async (ctx) => {
  const {a, b} = ctx.args;
  return a + b;
}
```

小程序端发起对云函数 `sum` 的调用示例如下。

```
const { result } = await mpserverless.function.invoke('sum', { a: 1, b: 1 });
console.log('1 + 1 = ', result);  // 1 + 1 = 2
```

在云函数端任意云函数发起对云函数 `sum` 的调用示例如下。

```
'use strict';

module.exports = async (ctx) => {
  const { result } = ctx.mpserverless.function.invoke('sum', { a: 1, b: 1 });
  return result;
}
```

2.2. 数据存储

2.2.1. aggregate

对数据库执行聚合查询。

方法定义

```
aggregate(pipeline: object[], options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
pipeline	Array	是	聚合查询对象。
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明

options.explain	Boolean	否	是否返回执行计划。
options.allowDiskUse	Boolean	否	是否在聚合查询执行的过程中使用磁盘存储临时数据。
options.bypassDocumentValidation	Boolean	否	是否允许绕过文档验证。
options.raw	Boolean	否	是否将结果作为BSON Buffer返回。
options.promoteLongs	Boolean	否	是否将long数据类型提升为Number。
options.promoteValues	Boolean	否	是否将BSON值的类型提升为本地类型。
options.promoteBuffers	Boolean	否	是否将BSON Buffer 转换为Node Buffer。
options.collation	Object	否	指定更新的排序顺序。

示例

- 不对数据进行分组，统计orders集合中所有total字段的和，并且要求在1000ms内返回结果否则报错。

```
mpserverless.db.collection('orders').aggregate([
  {
    $group:
    {
      _id : null,
      total:{$sum: "$total"}
    }
  }
 ], {
 maxTimeMS: 1000
})
```

- 查询2019年第一季度（1.1~3.31）已完成订单（status字段值为completed）的订单总金额（商品费用total和运费shippingFee之和）和订单总数，并且允许绕过文档验证。

```
mpserverless.db.collection('orders').aggregate([
 // 1.利用match匹配状态和日期条件
  {
   $match: {status: "completed", orderDate: {
     $gte: ISODate("2019-01-01"),
      $lt: ISODate("2019-04-01")
    }}
  },
  // 2.聚合订单总金额、总运费、总数量
  {
   $group:{
     _id: null,
      total: {$sum: "$total"},
      shippingFee: {$sum: "$shippingFee"},
      count:( $sum : 1)
    }
  },
  // 3.计算总金额
  {
   $project: {
     grandTotal: {$add: ["$total", "shippingFee"]},
      count: 1,
      _id: 0
    }
  }
], {
  bypassDocumentValidation: true,
})
```

2.2.2. count

获取集合中符合条件的记录数量。

方法定义

```
count(query: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
query	Object	是	数据库操作时的过滤条件。
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明
options.limit	Number	否	限制count的文档数量。
options.skip	Number	否	count前跳过的文档数量。

示例

- 统计所有存储在集合users中的记录的数量。并打印结果。

```
mpserverless.db.collection('users').count().then((res) => {
  console.log(res);
}).catch(console.error);
```

- 查找集合users中所有age大于等于18的记录数量，并打印结果。

```
mpserverless.db.collection('users').count({
  age: {$gt: 18}
}).then((res) => {
  console.log(res);
}).catch(console.error);
```

- 查找集合myNoteBook中title字段为1，userId字段为 `user._openid` 的记录数量。并打印结果。

```
mpserverless.db.collection('myNoteBook').count({
  title: "1",
  userId: user._openid,
}).then((res) => {
  console.log(res);
}).catch(console.error);
```

2.2.3. deleteOne

删除集合中的一条记录。

方法定义

```
deleteOne(filter: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明

字段名	类型	必填	说明
filter	Object	是	数据库操作时的过滤条件

示例

删除users集合中age小于18的第一条记录。

```
mpserverless.db.collection('users').deleteOne({age: {$lt: 18}).then((res) => {
  const hasDeleted = res.affectedDocs > 0;
}).catch(console.error);
```

2.2.4. deleteMany

删除集合中的一批记录。

方法定义

```
deleteMany(filter: object): Promise<MongoResult>;
```

请求参数

字段名	类型	必填	说明
filter	Object	是	数据库操作时的过滤条件。

示例

- 删除所有 users 集合中 age 字段值小于 18 的数据条目。

```
mpserverless.db.collection('users').deleteMany({age: {$lt: 18}}).then((res) => {
  const hasDeleted = res.affectedDocs > 0;
}).catch(console.error);
```

- 删除所有 users 集合中 age 字段值大于 18 且小于等于 60 的数据条目。

```
mpserverless.db.collection('users').deleteMany({age: {$gt: 18, $lte: 60}}).then((res) =>
{
  const hasDeleted = res.affectedDocs > 0;
}).catch(console.error);
```

2.2.5. distinct

获取某个属性去重后的所有记录。

方法定义

```
distinct(key: string, query: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
key	String	是	待获取的属性名。
query	Object	是	数据库操作时的查询条件。
options	Object	否	控制项。

示例

返回age字段大于18的所有姓名（如果有相同的name只返回一个）。

```
mpserverless.db.collection('users').distinct('name', {age: {$gt: 18}})
.then((res) => {})
.catch(console.error);
```

2.2.6. find

查找集合中符合条件的所有记录。

方法定义

```
find(query?: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
query	Object	否	数据库操作时的查询条件。
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明
options.limit	Number	否	查询的文档数量限制。
options.skip	Number	否	跳过的文档数量。
options.sort	Object	否	指定排序的字段，并使用1和-1来指定排序的方式。 其中： • 1：表示升序排列 • -1：表示降序排列
options.projection	Object	否	使用投影操作符指定返回的键，值设置为1的字段返回，值为0的字段隐藏。
options.hint	Object	否	指定查询时使用的索引

示例

- 在users集合中查找age字段值大于18的记录，按照姓名正序排列，返回符合条件的第11到第20条记录的name字段（_id字段默认也返回）。

```
mpserverless.db.collection('users').find({
    age: {$gt: 18}
}, {
    projection: {name: 1},
    sort: {name: 1},
    skip: 10,
    limit: 10,
})
.then(res => {})
.catch(console.error);
```

- 在users集合中对所有记录按照姓名正序排列，返回符合条件的第11到第20条记录的name字段（_id字段默认也返回）。

```
mpserverless.db.collection('users').find({}, {
    projection: {name: 1},
    sort: {name: 1},
    skip: 10,
    limit: 10,
})
.then(res => {})
.catch(console.error);
```

2.2.7. findOne

查询单条记录。

方法定义

```
findOne(query?: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
query	Object	否	数据库操作时的查询条件。
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明
options.skip	Number	否	跳过的文档数量。
options.sort	Object	否	指定排序的字段，并使用1和-1来指定排序的方式。 其中： • 1：表示升序排列 • -1：表示降序排列
options.projection	Object	否	使用投影操作符指定返回的键，值设置为1的字段返回，值为0的字段隐藏。
options.hint	Object	否	指定查询时使用的索引。

示例

- 在users集合中查找age字段值大于18的记录，按照姓名升序排列，跳过10条记录后，返回符合条件的第一条记录的name字段（_id字段也默认返回）。并且搜索时不使用_id字段作为索引，而是以name字段作为索引。

```
mpserverless.db.collection('users').findOne({
    age: {$gt: 18}
}, {
    projection: {name: 1},
    sort: {name: 1},
    skip: 10,
    hint: {name: 1}
})
.then(res => {})
.catch(console.error);
```

- 在users集合中按照姓名升序排列，跳过10条记录后，返回符合条件的第一条记录的name字段（_id字段也默认返回）。并且搜索时不使用_id字段作为索引，而是以name字段作为索引。

```
mpserverless.db.collection('users').findOne({}, {
    projection: {name: 1},
    sort: {name: 1},
    skip: 10,
    hint: {name: 1}
})
.then(res => {})
.catch(console.error);
```

2.2.8. findOneAndDelete

删除一条记录，并将这条记录返回。

方法定义

```
findOneAndDelete(filter: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
filter	Object	是	数据库操作时的查询条件。
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明

options.sort	Object	否	指定排序的字段，并使用 1 和 -1 来指定排序的方式。 其中： • 1：表示升序排列 • -1：表示降序排列
options.projection	Object	否	使用投影操作符指定返回的键，值设置为1的字段返回，值为0的字段隐藏。默认返回_id字段，如果不希望返回该字段需要额外指定。

示例

- 查询users集合中的age字段值等于18的一条记录，将其删除并返回该记录。如果有多条则返回并删除第一条。

```
mpserverless.db.collection('users')
.findOneAndDelete({
    age: 18,
})
.then(res => {})
.catch(console.error)
```

- 查询users集合中age字段值等于18的一条记录，将其删除并返回该记录的name字段。如果有多条则返回删除其name字段升序排列的第一个。

```
mpserverless.db.collection('users').findOneAndDelete({
 age: 18,
}, {
    projection: {_id: 0, name: 1},
    sort: {name: 1},
})
.then(res => {})
.catch(console.error);
```

2.2.9. findOneAndReplace

整体替换一条记录，并返回替换前的记录。

方法定义

```
findOneAndReplace(filter: object, replacement: object, options?: object): Promise<MongoResu
lt>
```

请求参数

字段	类型	必填	描述
filter	Object	是	数据库操作时的查询条件。
replacement	Object	是	数据库操作时的替换对象。
options	Object	否	控制项。

options参数定义：

字段	类型	必填	描述
options.sort	Object	否	排序规则
options.upsert	Boolean	否	如果查找不到对应文档，是否插入。默认值：false。
options.projection	Object	否	查询后过滤的字段。

示例

- 从clubs集合中查找score字段值大于20000的第一条记录，并将其替换为 `{ team: "Therapeutic Hamsters", score: 22250}` 。

```
mpserverless.db.collection("clubs")
  .findOneAndReplace({
     "score": { $gt: 20000 }
  },
  {
     "team": "Therapeutic Hamsters",
     "score": 22250
  })
  .then(res => {})
  .catch(console.error);
```

- 在users集合中查找age字段值大于18的按照姓名升序排列的第一条记录，返回该记录的name字段（_id字段也会返回）。同时将该记录修改为 `{name: "Smith",age: 22}` 。若没有找到符合条件的记录，就将 `{name: "Smith",age: 22}` 作为新记录插入。

```
mpserverless.db.collection("users").findOneAndReplace({
    age: {$gt: 18}
}, {
  name: "Smith",
  age: 22,
}, {
    projection: {name: 1},
    sort: {name: 1},
    upsert: true,
})
.then(res => {})
.catch(console.error);
```

2.2.10. findOneAndUpdate

查询并整体更新一条记录。

方法定义

```
findOneAndUpdate(filter: object, update: object, options?: object): Promise<MongoResult>
```

请求参数

字段	类型	是否必选	描述
filter	Object	是	数据库操作时的查询条件。
update	Object	是	数据库操作时的替换对象。
options	Object	否	控制项。

options参数定义：

字段	类型	是否必选	描述
options.sort	Object	否	排序规则。
options.upsert	Boolean	否	如果查找不到对应文档，是否插入。默认值：false。

字段	类型	是否必选	描述
options.projection	Object	否	查询后过滤的字段。 • 0：表示不返回。 • 1：表示返回。
options.returnNewDocument	Boolean	否	是否返回修改后对象取值。 • true：表示返回更新后的文档。 • false（默认）：表示返回更新前的文档。

示例

- 从clubs集合中查找score字段值大于20000的第一条记录，并将其team字段设置为Therapeutic Hamsters，score字段设置为22250，其他字段保持不变。

```
mpserverless.db.collection("clubs")
  .findOneAndUpdate({
     "score": { $gt: 20000 }
  }, {
   $set: {
     "team": "Therapeutic Hamsters",
     "score": 22250,
   }
  })
  .then(res => {})
  .catch(console.error);
```

- 在users集合中查找age字段值大于18的按照姓名升序排列的第一条记录，返回该记录的name字段（_id字段也会返回）。同时将该记录的name和age字段更新，其他字段默认保留。若没有找到符合条件的记录，就将 `{ name: "Smith",age: 22 }` 作为新记录插入。

```
mpserverless.db.collection("users").findOneAndUpdate({
    age: {$gt: 18}
}, {
  $set: {
  name: "Smith",
  age: 22,
  }
}, {
    projection: {name: 1},
    sort: {name: 1},
    upsert: true,
})
.then(res => {})
.catch(console.error);
```

2.2.11. replaceOne

查询并整体替换这条记录。

方法定义

```
replaceOne(filter: object, doc: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
filter	Object	是	数据库操作时的过滤条件。
doc	Object	是	替换的目标文档。
options	Object	否	控制项。

options参数定义：

字段	类型	必填	描述
options.upsert	Boolean	否	如果查找不到对应文档，是否插入。默认值：false。

示例

- 在users集合中查询name字段值为tom、age字段值为18的第一条记录，并将该记录替换为 `{ name: 'jerry' }` 。如果没有符合条件的记录，不作任何操作。

```
mpserverless.db.collection('users')
.replaceOne({
    name: 'tom'
    age: 18
}, {
    name: 'jerry'
})
.then(res => {})
.catch(console.error);
```

- 在users集合中查询name字段值为tom、age字段值为18的第一条记录，并将该记录替换为 `{ name: 'jerry' }` 。如果没有符合条件的记录，就插入一个新记录。

```
mpserverless.db.collection('users')
.replaceOne({
    name: 'tom'
    age: 18
}, {
    name: 'jerry'
}, {
  upsert: true
})
.then(res => {})
.catch(console.error);
```

2.2.12. insertMany

在集合中添加一批记录。

方法定义

```
insertMany(docs: object[]): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
docs	Array	是	待插入的数据，只能为对象数组，数组中的每个元素作为一个数据条目。docs 字段不能为空数组 [] 。

示例

向users集合中插入两条记录。

```
mpserverless.db.collection('users').insertMany([{
    name: 'tom',
    age: 1
}, {
    name: 'jerry',
    age: 2
}])
.then(res => {})
.catch(console.error)
```

2.2.13. insertOne

在集合中添加一条记录。

方法定义

```
insertOne(doc: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
doc	Object	是	表示即将被插入的元素。

示例

向users集合中插入一条数据。

```
mpserverless.db.collection('users').insertOne({
    name: 'tom',
    age: 1
})
.then(res => {})
.catch(console.error)
```

2.2.14. updateMany

更新集合中的一批记录。

方法定义

```
updateMany(filter: object, update: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
filter	Object	是	过滤条件。
update	Object	是	更新的文档。
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明

options.upsert	Boolean	否	不匹配时是否直接插入文档。默认值：false。

示例

- 在users集合中查找所有name字段值为jerry的记录，并将其age字段更新为10。

```
mpserverless.db.collection('users').updateMany({
    name: 'jerry'
}, {
    $set: {
        age: 10
    }
})
.then(res => {})
.catch(console.error)
```

- 在users集合中查找所有age字段值大于18 的记录，同时将该记录的name字段和age字段更新，其他字段默认保留。若没有找到符合条件的记录，就将 `{name: "Smith",age: 22}` 作为新记录插入。

```
mpserverless.db.collection('users').updateMany({
    age: {$gt: 18}
}, {$set: {
    name: "Smith",
    age: 22,
    }
}, {
    upsert: true,
})
.then(res => {})
.catch(console.error);
```

2.2.15. updateOne

更新一条记录。

方法定义

```
updateOne(filter: object, update: object, options?: object): Promise<MongoResult>
```

请求参数

字段名	类型	必填	说明
filter	Object	是	过滤条件。
update	Object	是	更新的文档。

字段名	类型	必填	说明
options	Object	否	控制项。

options参数定义：

字段名	类型	必填	说明
options.upsert	Boolean	否	不匹配时是否直接插入文档。默认值：false。

示例

- 在users集合中查找name字段值为jerry的第一条记录，并将其age字段更新为10。

```
mpserverless.db.collection('users').updateOne({
    name: 'jerry'
}, {
    $set: {
        age: 10
    }
})
.then(res => {})
.catch(console.error)
```

- 在users集合中查找age字段值大于18的第一条记录，同时将该记录的name和age字段更新，其他字段默认保留。若没有找到符合条件的记录，就将 `{name: "Smith",age: 22}` 作为新记录插入。

```
mpserverless.db.collection('users').updateOne({
    age: {$gt: 18}
}, {$set: {
    name: "Smith",
    age: 22,
    }
}, {
    upsert: true,
})
.then(res => {})
.catch(console.error);
```

2.3. 文件存储

2.3.1. uploadFile

上传文件。

方法定义

```
mpserverless.file.uploadFile(options: object): Promise<Result>
```

请求参数

字段名	类型	必填	说明
options.filePath	String	是	本地文件路径，通常可以从图片文件描述中获取文件路径。可上传文件大小限制在 100 MB 以内。
options.fileSize	Number	否	上传的文件大小。
options.extension	String	否	上传的文件的扩展名。目前支持上传以下格式的文件： • .jpg • .jpeg • .png • .gif • .image
options.env	String	否	文件的获取方式。 唯一可选值 public： 可公开访问的文件。
options.timeout	Number	否	超时时间，以毫秒为单位。默认值：60000。
options.headers	Object	否	文件响应头键值对，可定义如下内容： • cacheControl： 缓存策略。 • contentDisposition：文件形式。 • contentEncoding： 文件内容编码。 • expires： 缓存有效时长。

options.meta	Object	否	自定义文件响应头键值对。例如，自定义 userId: halo 获得响应头 x-meta-user-id: halo 。

返回参数

字段名	类型	说明
filePath	String	本地文件路径。
fileUrl	String	上传文件后获得的文件链接地址。

示例

- 支付宝小程序上传文件示例。

```
my.chooseImage({
  chooseImage: 1,
  success: res => {
    const path = res.apFilePaths[0];
    const options = {
      filePath: path,
    };

    mpserverless.file.uploadFile(options)
    .then(res => { // do somthing  })
    .catch(err => { // deal error  });
  },
});
```

- 微信小程序上传文件示例。

```
wx.chooseImage({
  count: 1,
  sizeType: ['original', 'compressed'],
  sourceType: ['album', 'camera'],
  success: res => {
    const path = res.tempFilePaths[0];
    const options = {
      filePath: path,
    };

    mpserverless.file.uploadFile(options)
    .then(res => { // do somthing  })
    .catch(err => { // deal error  });
  },
});
```

- 钉钉小程序上传文件示例。

```
dd.chooseImage({
  chooseImage: 1,
  success: res => {
    const path = res.apFilePaths[0];
    const options = {
      filePath: path,
    };

    mpserverless.file.uploadFile(options)
    .then(res => { // do somthing  })
    .catch(err => { // deal error  });
  },
});
```

2.3.2. deleteFile

删除已上传的文件。

方法定义

```
mpserverless.file.deleteFile(url: string): Promise<Result>
```

请求参数

字段名	类型	必填	说明
url	String	是	文件存储完整路径

返回参数

字段名	类型	说明
success	Boolean	执行状态。
result	any	

示例

假设已知一个上传文件后获得的文件链接地址 `fileURL` ，根据文件地址从服务空间中删除该文件。

```
const fileURL = 'https://resource.bspapp.com/xxx-xx/4b82ded0-0118-4de4-9f50-ab13110a1ffb.jpg';

mpserverless.file.deleteFile(fileURL).then(res => {
  console.log(res);
}).catch(err => {
  console.error(err);
});
```

2.4. 用户管理

2.4.1. getInfo

获取用户信息。

方法定义

```
mpserverless.user.getInfo(options: object): Promise<Result>
```

请求参数

字段名	类型	必填	说明
options	Object	否	`authProvider{String}` 指定客户端类型，取值： • alipay_openapi：支付宝（默认值） • wechat_openapi：微信 • dingtalk_openapi：钉钉

返回参数

字段名	类型	说明
success	Boolean	操作是否成功。

字段名	类型	说明
result	Object	用户信息： • `spaceId{String}`：服务空间ID。 • `userId{String}`：用户ID。 • `oAuthUserId{String}`：三方授权用户标识。

示例

在调用 getInfo 接口之前，请先完成 SDK 的初始化。

```
await mpserverless.init(); // 初始化

const {result} = await mpserverless.user.getInfo();
console.log(result) // { spaceId: xxx, userId: xxx, oAuthUserId: xxxx}
```

3.客户端API文档（2.x）

3.1. 云函数

3.1.1. invoke

调用云服务。

方法定义

该方法的定义如下：

```
mpserverless.function.invoke(functionName: string, options?: object): Promise<Result>
```

返回参数

字段名	类型	说明
success	Boolean	执行状态。
requestId	String	请求ID。
result	Any	接口返回内容，由开发者代码和请求参数 header 的 content-type 决定，默认 content-type 为 application/json。

示例

```
mpserverless.function.invoke('dataAnalytics', {
    range: 30,
}).then((res) => {
  console.log(res);
}).catch(console.error);
```

3.2. 数据存储

3.2.1. aggregate

对数据库执行聚合查询。

方法定义

该方法的定义如下：

```
aggregate(pipeline: object | object[], options?: object): Promise<MongoResult>
```

请求参数

该方法接收11个参数，其定义如下：

字段名	类型	必填	说明
pipeline	Array	是	聚合查询对象。
options	Object	否	控制项。
options.explain	Boolean	否	是否返回执行计划。
options.allowDiskUse	Boolean	否	是否在聚合查询执行的过程中使用磁盘存储临时数据。
options.maxTimeMS	Number	否	执行查询的超时时间。
options.bypassDocumentValidation	Boolean	否	是否允许绕过文档验证。
options.raw	Boolean	否	是否将结果作为 BSON Buffer 返回。
options.promoteLongs	Boolean	否	是否将 long 数据类型提升为 Number。
options.promoteValues	Boolean	否	是否将 BSON 值的类型提升为本地类型。
options.promoteBuffers	Boolean	否	是否将 BSON Buffer 转换为 Node Buffer。
options.collation	Object	否	指定更新的排序顺序。

示例

```
mpserverless.db.collection('users').aggregate([
    {
      $match: {
        _id: {
          $in: insertRet.insertedIds
        }
      },
    },
    {
      $project: {
        name: 1,
        address: 1
      }
    }, {
      $group: {
        _id: '$name',
        count: {$sum: 1}
      }
    },{
      $match: {
        count: {
          $gt: 1
        }
      }
    }
  ]);
```

3.2.2. count

获取集合中符合条件的记录数量。

方法定义

该方法的定义如下：

```
count(query: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 5 个参数，其定义如下：

字段名	类型	必填	说明
query	Object	否	数据库操作时的过滤条件。
options	Object	否	控制项。
options.limit	Boolean	否	是否限制 count 的文档数量。

字段名	类型	必填	说明
options.skip	Number	否	count 前跳过的文档数量。
options.maxTimeMS	Number	否	超时时间。

示例

```
mpserverless.db.collection('users').count({age: {$gt: 18}}).then((res) => {
    console.log(res);
}).catch(console.error);
```

3.2.3. deleteOne

删除集合中的一条记录。如果没有查询条件，则默认删除第一行。

方法定义

该方法的定义如下：

```
deleteOne(filter: object): Promise<MongoResult>
```

请求参数

该方法接收一个参数，其定义如下：

字段名	类型	必填	说明
filter	Object	否	数据库操作时的过滤条件

示例

```
mpserverless.db.collection('users').deleteOne({age: {$lt: 18}}).then((res) => {
  const hasDeleted = res.affectedDocs > 0;
}).catch(console.error);
```

3.2.4. deleteMany

删除集合中的一批记录。

方法定义

该方法的定义如下：

```
deleteMany(filter: object): Promise<MongoResult>;
```

请求参数

字段名	类型	必填	说明
filter	Object	是	数据库操作时的过滤条件。

示例

```
mpserverless.db.collection('users').deleteMany({age: {$lt: 18}}).then((res) => {
  const hasDeleted = res.affectedDocs > 0;
}).catch(console.error);
```

3.2.5. distinct

获取某个属性去重后的所有记录。

方法定义

该方法的定义如下：

```
distinct(key: string, query: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 3 个参数，其定义如下：

字段名	类型	必填	说明
key	String	是	待获取的属性名。
query	Object	是	数据库操作时的查询条件。
options	Object	否	控制项。

示例

```
mpserverless.db.collection('users').distinct('name', {age: {$gt: 18}})
.then((res) => {})
.catch(console.error);
```

3.2.6. find

查找集合中符合条件的所有记录。

方法定义

该方法的定义如下：

```
find(query?: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 8 个参数，其定义如下：

字段名	类型	必填	说明
query	Object	否	数据库操作时的查询条件。
options	Object	否	控制项。
options.limit	Number	否	查询的文档数量限制。
options.skip	Number	否	跳过的文档数量。
options.maxTimeMS	Number	否	超时时间。
options.sort	Object	否	指定排序的字段，并使用 1 和 -1 来指定排序的方式。其中： ● 1：表示升序排列 ● -1：表示降序排列
options.projection	Object	否	使用投影操作符指定返回的键，值设置为1的字段返回，值为0的字段隐藏。
options.hint	Object	否	指定查询时使用的索引。

示例

```
mpserverless.db.collection('users').find({
    age: {$gt: 18}
}, {
    projection: {name: 1},
    limit: 10,
    skip: 10,
    sort: {name: 1},
    hint: {name: 1, _id: 0}
})
.then(res => {})
.catch(console.error);
```

3.2.7. findOne

查询单条记录。

方法定义

该方法的定义如下：

```
findOne(query?: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 8 个参数，其定义如下：

字段名	类型	必填	说明
query	Object	否	数据库操作时的查询条件。
options	Object	否	控制项。
options.limit	Number	否	查询的文档数量限制。
options.skip	Number	否	跳过的文档数量。
options.maxTimeMS	Number	否	超时时间。
options.sort	Object	否	指定排序的字段，并使用 1 和 -1 来指定排序的方式。其中： ● 1：表示升序排列 ● -1：表示降序排列
options.projection	Object	否	使用投影操作符指定返回的键，值设置为1的字段返回，值为0的字段隐藏。
options.hint	Object	否	指定查询时使用的索引。

示例

```
mpserverless.db.collection('users')
.findOne({
    age: {$gt: 18}
}, {
    projection: {name: 1},
    limit: 10,
    skip: 10,
    sort: {age:1}
})
.then(res => {})
.catch(console.error)
```

3.2.8. findOneAndDelete

查询并删除一条记录。

方法定义

该方法的定义如下：

```
findOneAndDelete(query?: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 5 个参数，其定义如下：

字段	类型	必填	说明
query	Object	是	数据库操作时的查询条件。
options	Object	否	控制项。
options.maxTimeMS	Number	否	超时时间。
options.sort	Object	否	排序规则。
options.projection	Object	否	查询后过滤的字段。

示例

```
mpserverless.db.collection('users')
.findOneAndDelete({
    age: 18
})
.then(res => {})
.catch(console.error)
```

3.2.9. findOneAndReplace

查询并整体替换一条记录。

方法定义

该方法的定义如下：

```
findOneAndReplace(filter: object, replacement: object, options?: object): Promise<MongoResu
lt>
```

请求参数

该方法接收 6 个参数，其定义如下：

字段	类型	必填	描述
filter	Object	是	数据库操作时的查询条件。
replacement	Object	是	数据库操作时的替换对象。
options	Object	否	控制项。
options.maxTimeMS	Number	否	超时时间。

字段	类型	必填	描述
options.sort	Object	否	排序规则
options.upsert	Boolean	否	如果查找不到对应文档，是否插入。默认值：*false*。
options.projection	Object	否	查询后过滤的字段。

示例

```
mpserverless.db.collection(database)
  .findOneAndReplace({
     "score": { $gt: 20000 }
  },
  {
     "team": "Therapeutic Hamsters",
     "score": 22250
  })
  .then(res => {})
  .catch(console.error);
```

3.2.10. findOneAndUpdate

查询并更新记录。

方法定义

该方法的定义如下：

```
findOneAndUpdate(filter: object, update: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 7个参数，其定义如下：

字段	类型	必填	说明
filter	Object	是	数据库操作时的过滤条件。
update	Object	是	数据库操作时的更新对象。
options	Object	否	控制项。
options.maxTimeMS	Number	否	超时时间。
options.sort	Object	否	排序规则。

字段	类型	必填	说明
options.upsert	Boolean	否	如果查找不到对应文档，是否插入。默认值：*false*。
options.projection	Object	否	查询后过滤的字段。

示例

```
mpserverless.db.collection('users')
    .findOneAndUpdate({
        username: "zhangsan"
    },
    {
    $set: {
        age: 18
        }
    })
    .then(res => {})
    .catch(console.error)
```

3.2.11. replaceOne

查询并整体替换这条记录。

方法定义

该方法的定义如下：

```
replaceOne(filter: object, doc: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 4 个参数，其定义如下：

字段名	类型	必填	说明
filter	Object	是	数据库操作时的过滤条件。
doc	Object	是	替换的目标文档。
options	Object	否	控制项。
options.upsert	Boolean	否	如果匹配不到，是否插入。默认值：*false*。

示例

```
mpserverless.db.collection('users')
.replaceOne({
    name: 'tom'
    age: 18
},{
    name: 'jerry'
})
.then(res => {})
.catch(console.error);
```

3.2.12. insertMany

在集合中添加一批记录。

方法定义

该方法的定义如下：

```
insertMany(docs: object[]): Promise<MongoResult>
```

请求参数

该方法接收 1 个参数，其定义如下：

字段名	类型	必填	说明
docs	Array	是	待插入的数据，只能为数组。

示例

```
mpserverless.db.collection('users').insertMany([{
    name: 'tom',
    age: 1
},{
    name: 'jerry',
    age: 2
}])
.then(res => {})
.catch(console.error)
```

3.2.13. insertOne

在集合中添加一条记录。

方法定义

该方法的定义如下：

```
insertOne(doc: object): Promise<MongoResult>
```

请求参数

该方法接收 1 个参数，其定义如下：

字段名	类型	必填	说明
doc	Object	是	待插入的数据

示例

```
mpserverless.db.collection('users').insertOne({
    name: 'tom',
    age: 1
})
.then(res => {})
.catch(console.error)
```

3.2.14. updateMany

更新集合中的一批记录。

方法定义

该方法的定义如下：

```
updateMany(filter: object, update: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 4 个参数，其定义如下：

字段名	类型	必填	说明
filter	Object	是	过滤条件。
update	Object	是	更新的文档
options	Object	否	控制项
options.upsert	Boolean	否	不匹配时是否直接插入文档。默认值：false。

示例

```
mpserverless.db.collection('users').updateMany({
    name: 'jerry'
}, {
    $set: {
        age: 10
    }
})
.then(res => {})
.catch(console.error)
```

3.2.15. updateOne

更新集合中的一条记录。

方法定义

该方法的定义如下：

```
updateOne(filter: object, update: object, options?: object): Promise<MongoResult>
```

请求参数

该方法接收 4 个参数，其定义如下：

字段名	类型	必填	说明
filter	Object	是	过滤条件。
update	Object	是	更新的文档。
options	Object	否	控制项。
options.upsert	Boolean	否	不匹配时是否直接插入文档。默认值：*false*。

示例

```
mpserverless.db.collection('users').updateOne({
    name: 'jerry'
}, {
    $set: {
        age: 10
    }
})
.then(res => {})
.catch(console.error)
```

3.3. 文件存储

3.3.1. uploadFile

上传文件。

方法定义

该方法的定义如下：

```
mpserverless.file.uploadFile(options: object): Promise<Result>
```

请求参数

该方法接收 6 个参数，其定义如下：

字段名	类型	必填	说明
options	Object	是	上传控制项。
options.fileSize	String	否	上传的文件大小。
options.extension	String	否	上传的文件的扩展名。 目前支持上传以下格式的文件： • .jpg • .jpeg • .png • .gif • .image
options.filePath	String	是	本地文件路径，通常可以从图片文件描述中获取文件路径。可上传文件大小限制在 100 MB 以内。
options.env	String	否	文件的获取方式。 唯一可选值 *public*： 可公开访问的文件。
options.timeout	Number	否	超时时间，以毫秒为单位。默认值：*60000*。
options.headers	Object	否	文件响应头键值对，可定义如下内容： • *cacheControl*： 缓存策略。 • *contentDisposition*： 文件形式。 • *contentEncoding*： 文件内容编码。 • *expires*： 缓存有效时长。
options.meta	Object	否	自定义文件响应头键值对。例如，自定义 userId: halo 获得响应头 x-meta-user-id: halo 。

返回参数

字段名	类型	说明
filePath	String	本地文件路径。
fileUrl	String	上传文件后获得的文件链接地址。

示例

小程序选取文件上传示例。

```
my.chooseImage({
  chooseImage: 1,
  success: res => {
    const path = res.apFilePaths[0];
    const options = {
      filePath: path,
      headers: {
        contentDisposition: 'attachment',
      },
    };
    mpserverless.file.uploadFile(options).then(console.log).catch(console.error);
  },
});
```

3.3.2. deleteFile

删除之前上传的文件。

方法定义

该方法的定义如下：

```
mpserverless.file.deleteFile(url: string): Promise<Result>
```

请求参数

该方法接收 1 个参数，其定义如下：

字段名	类型	必填	说明
url	String	是	线上完整路径

示例

```
mpserverless.file.deleteFile('https://resource.bspapp.com/xxx-xx/4b82ded0-0118-4de4-9f50-ab
13110a1ffb.jpg').then(res => {
  console.log(res);
}).catch(err => {
  console.error(err);
});
```

3.4. 用户管理

3.4.1. getInfo

获取用户信息。

方法定义

该方法的定义如下：

```
mpserverless.user.getInfo(options: object): Promise<Result>
```

请求参数

该方法接收以下请求参数。

字段名	类型	必填	说明
options	Object	否	authProvider{String} 指定客户端类型，取值： • *alipay_openapi*：支 付宝（默认 值） • *wechat_openapi*：微信 • *dingtalk_openapi*：钉钉

返回参数

字段名	类型	说明
success	Boolean	操作是否成功。
result	Object	用户信息： • spaceId{String}：服务空间ID。 • userId{String}：用户ID。 • oAuthUserId{String}：三方授权用户标识。只有指定了authProvider 时才会返回该参数。

示例

- 匿名授权方式获取用户信息。

```
const res = await mpserverless.user.authorize({   authType:'anonymous' }); if (res.succes
s) {   console.log('授权成功'); } mpserverless.user.getInfo().then(user => { this.setData(
userInfo: user); }).catch(console.error);
```

- 非匿名授权方式获取用户信息。
  - 支付宝授权：

```
const res = await mpserverless.user.authorize({ authProvider: 'alipay_openapi' }); if (
res.success) { console.log('授权成功'); } mpserverless.user.getInfo().then(user => { thi
s.setData(userInfo: user); }).catch(console.error);
```

- 微信授权：

```
const res = await mpserverless.user.authorize({ authProvider: 'wechat_openapi' }); if (res.success) { console.log('授权成功'); } mpserverless.user.getInfo().then(user => { this.setData(userInfo: user); }).catch(console.error);
```

- 钉钉授权：

```
const res = await mpserverless.user.authorize({ authProvider: 'dingtalk_openapi' }); if (res.success) { console.log('授权成功'); } mpserverless.user.getInfo().then(user => { this.setData(userInfo: user); }).catch(console.error);
```

说明

- 应用获取授权后方能获取用户信息，获取授权接口信息参见authorize。
- 微信小程序中的授权和获取用户信息操作只能进行一次，建议开发者在app.js中授权并获取用户信息后，将用户信息保存到全局变量中，以便页面使用。

3.4.2. authorize

获取用户授权。

方法定义

该方法的定义如下：

```
authorize(options: AuthorizeOptions): Promise<{success: boolean;}>
```

请求参数

字段名	类型	必填	说明
options	Object	否	• authProvider{String}：客户端类型。取值： ○ *alipay_openapi*（默认值） ○ *wechat_openapi* • authType{String}：授权方式。取值： ○ 空字符串（默认值）。 ○ *anonymous*：匿名方式

说明

- 如果同时指定了authProvider和authType，authType配置优先。
- 如果将authType设置为anonymous，则不需要设置authProvider。
- 如果没有设置authType，则必须设置authProvider。

返回参数

字段名	类型	说明
success	Boolean	操作是否成功。

示例

- 非匿名授权：

```
const res = await mpserverless.user.authorize({
  authProvider: 'alipay_openapi'
});
if (res.success) {
  console.log('授权成功');
}
```

- 匿名授权：

```
const res = await mpserverless.user.authorize({
  authType:'anonymous'
});
if (res.success) {
  console.log('授权成功');
}
```

说明 匿名授权获取的用户身份是一个临时身份，每次匿名授权得到的用户标识是随机不固定的，若需要获取固定用户标识，则需要通过三方授权获取。