云开发

开发指南

产品文档

腾讯云

文档目录

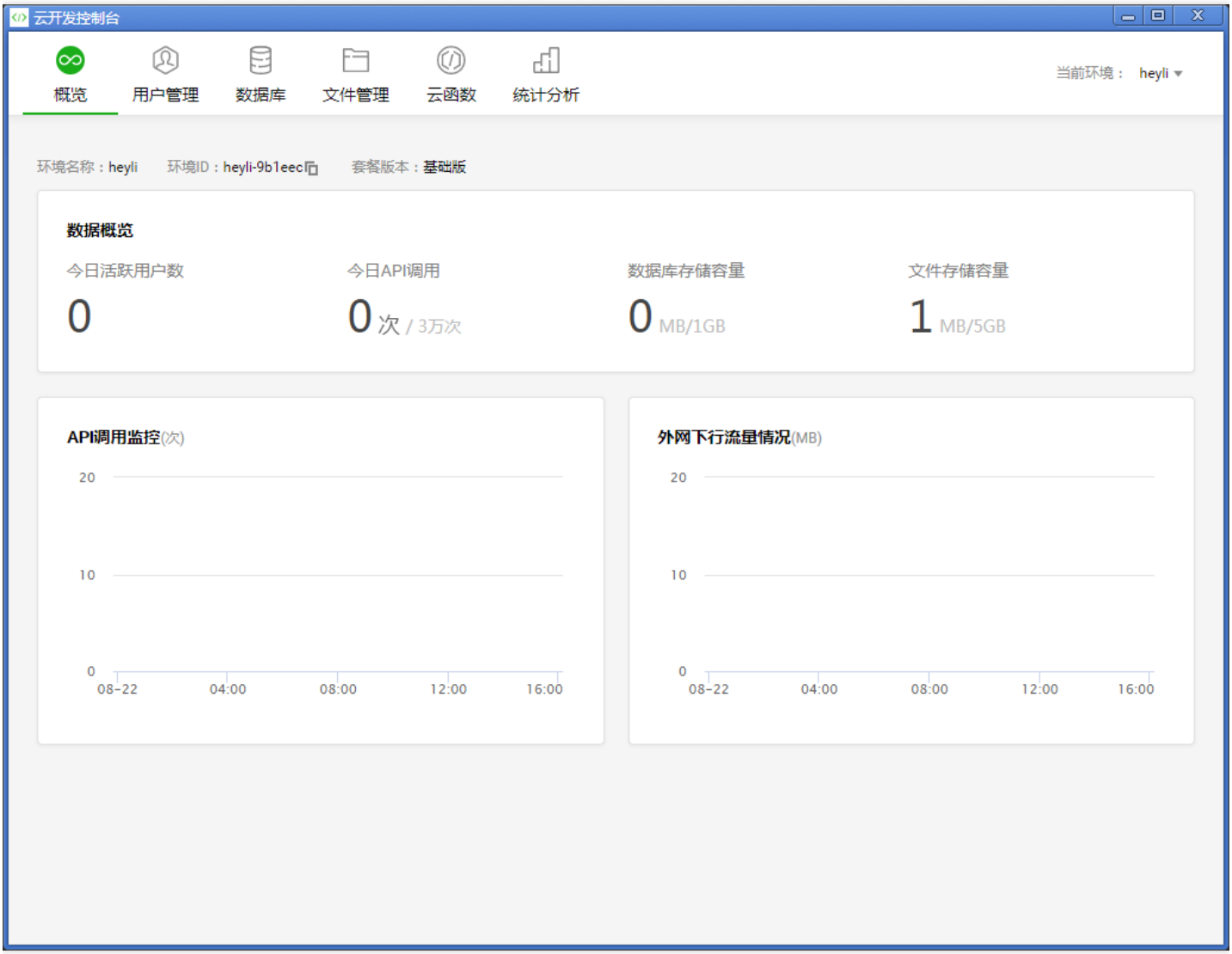
开发指南
小程序
云开发能力

最近更新时间：2018-09-11 16:33:45

云开发控制台

云开发控制台是管理云开发资源的地方，控制台提供以下能力：

- 概览：查看云开发基础使用数据
- 用户管理：查看小程序用户信息
- 数据库：管理数据库，可查看、增加、更新、查找、删除数据、管理索引、管理数据库访问权限等
- 文件管理：查看和管理存储空间
- 云函数：查看云函数列表、配置、日志和监控
- 统计分析：查看云开发资源具体使用统计信息



以上的能力当中，数据库、文件管理与云函数，是可以通过小程序的API接口或者服务端 SDK 进行调用的，下一节会进一步介绍。

小程序（客户端）使用云开发能力

客户端，这里是指在小程序端中。如果要使用云开发能力，请做以下配置：

- 在 `app.json / game.json` 中， 中增加字段 `"cloud": true`
- project.config.json 中增加了字段 cloudfunctionRoot 用于指定存放云函数的目录
- 初始化云开发能力：

```
//app.js
App({
```

```
onLaunch: function () {
wx.cloud.init({
traceUser: true
});
}
});
```

初始化能力文档

在用户管理中会显示使用云能力的小程序的访问用户列表，默认以访问时间倒叙排列，访问时间的触发点是在小程序端调用 `wx.cloud.init` 方法，且其中的 `traceUser` 参数传值为 `true`。

小程序操作文件资源

```
// 选择图片
wx.chooseImage({
success: dRes => {
// 上传图片
const uploadTask = wx.cloud.uploadFile({
cloudPath: `${Date.now()}-${Math.floor(Math.random(0, 1) * 10000000)}.png`, // 随机图片名
filePath: dRes.tempFilePaths[0], // 本地的图片路径
success: console.log,
fail: console.error
});
},
fail: console.error,
});
```

存储文档

小程序操作数据库

```
const db = wx.cloud.database();

db.collection('blog').get().then((res) => {
let data = res.data;
console.log(data);
});
```

数据库文档

小程序调用云函数

```
wx.cloud.callFunction({
name: 'addblog', // 云函数名称
```

```
data: { // 传到云函数处理的参数
title: '云开发 TCB',
content: '存储、数据库存、云函数'
}
}).then(res => {
console.log(res)
}).catch((err) => {
console.error(err);
});
```

云函数文档

服务端使用云开发能力

如果你想在云函数中，操作文件、数据库和云函数资源，你可以使用我们提供的服务端 SDK 进行操作。首先，进入到你的某个云函数中，安装以下依赖包：

```
npm i --save tcb-admin-node
```

在云函数中初始化

```
// 初始化示例
const app = require('tcb-admin-node');

// 初始化资源
// 云函数下不需要secretId和secretKey。
// env如果不指定将使用默认环境
app.init({
secretId: 'xxxxx',
secretKey: 'xxxx',
env: 'xxx'
});

//云函数下使用默认环境
app.init()

//云函数下指定环境
app.init({
env: 'xxx'
});
```

初始化文档

服务端操作文件资源

```
const app = require('tcb-admin-node');
app.init();

app.uploadFile({
cloudPath: "cover.png",
fileContent: fs.createReadStream(`${__dirname}/cover.png`)
}).then((res) => {
console.log(res);
}).catch((err) => {
console.error(err);
});;
```

存储文档

服务端操作数据库

```
const app = require('tcb-admin-node');
app.init();
const db = app.database();

db.collection('blogs').limit(10).get().then((res) => {
console.log(res);
}).catch((err) => {
console.error(err);
});;
```

数据库文档

服务端调用云函数

```
const app = require("tcb-admin-node");
app.init();

app.callFunction({
name: 'addblog', // 云函数名称
data: { // 传到云函数处理的参数
title: '云开发 TCB',
content: '存储、数据库存、云函数'
}
}).then((res) => {
console.log(res);
}).catch((err) => {
```

```
console.error(err);
});
```

云函数文档

通过本章，相信你已经知道如何在小程序端和服务端使用 SDK 去操作各类云资源。接下来，我们会分别详情讲述云函数、数据库和存储详细的使用方法。

语法糖

大部份的接口，目前都支持两种写法，分别是 Promise 和 Async/Await，本节以 `callFunction` 作为例子，在 `云函数` 中介绍这两种写法。 `Async/Awai t` 本质上是基于 `Promise` 的一种语法糖，它只是把 `Promise` 转换成同步的写法而已。

Promise

```
const app = require("tcb-admin-node");
app.init();

exports.main = (event, context, callback) => {
app.callFunction({
name: 'addblog', // 云函数名称
data: { // 传到云函数处理的参数
title: '云开发 TCB',
content: '存储、数据库存、云函数'
}
}).then((res) => {
console.log(res);
callback(null, res.data);
}).catch((err) => {
callback(err);
});
};
```

Async/Await

```
const app = require("tcb-admin-node");
app.init();

exports.main = async (event, context) => {
let result = null;

try {
```

```
result = await app.callFunction({
name: 'addblog', // 云函数名称
data: { // 传到云函数处理的参数
title: '云开发 TCB',
content: '存储、数据库存、云函数'
}
});
}
catch (e) {
return e;
}

return result;
};
```

你的第一个云开发小程序

最近更新时间：2018-09-11 16:45:46

什么是小程序·云开发

小程序·云开发是基于腾讯云的 云开发 TCB 的一站式小程序后端云服务。

开发者可以使用云开发开发微信小程序、小游戏，无需搭建服务器，即可使用云端能力。

云开发为开发者提供完整的云端支持，弱化后端和运维概念，无需搭建服务器，使用平台提供的 API 进行核心业务开发，即可实现快速上线和迭代，同时这一能力，同开发者已经使用的云服务相互兼容，并不互斥。

目前提供三大基础能力支持：

- 云函数：在云端运行的代码，微信私有协议天然鉴权，开发者只需编写自身业务逻辑代码。
- 数据库：一个既可在小程序前端操作，也能在云函数中读写的 JSON 数据库。
- 存储：在小程序前端直接上传/下载云端文件，在云开发控制台可视化管理。

用小程序·云开发制作你的博客小程序

下载或 clone 代码仓库

```
git clone https://github.com/TencentCloudBase/tcb-demo-blog.git
```

启动项目

用微信开发者工具，打开上一步下载下来的代码仓库，填入小程序的 AppID（使用云开发能力必须填写 AppID）。

> **注意**：
> 现在 2.2.3 或以上的基础库没有完全覆盖所有用户（目前约 90% ），如需使上传的代码能够覆盖全量用户，请做以下特殊处理：

1. 在 app.json / game.json 中增加字段 "cloud": true
2. 添加 `https://servicewechat.com` 为 request 可信域名，添加 `https://tcb-api.tencentcloudapi.com` 为 uploadFile 和 downloadFile 的可信域名

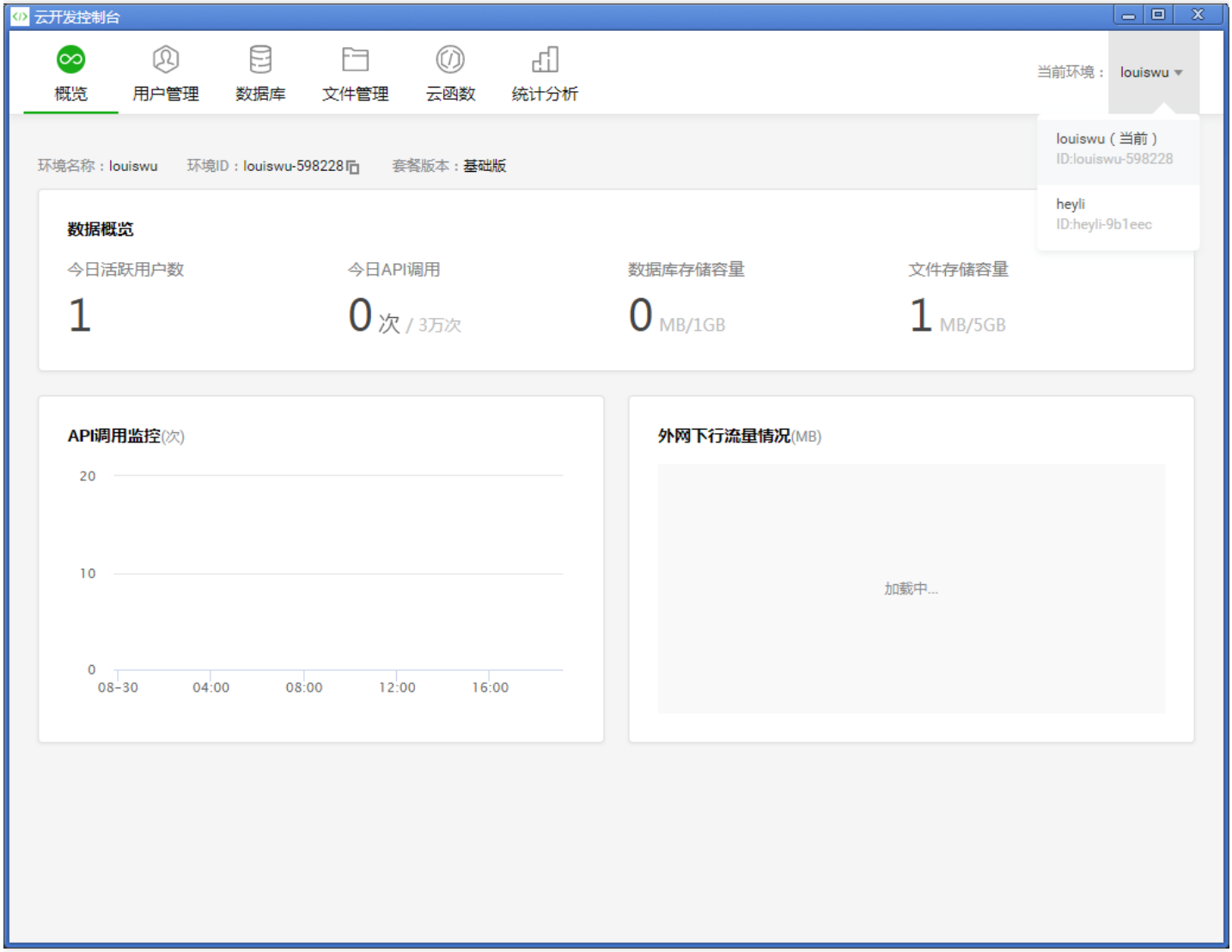
经过处理后会默认打入基础库 2.2.3 中附带的云能力，并且如果云能力有更新，并不会随着基础库升级而自动升级，需在后续版本发布后重新上传。如 2.2.4 发布后，需重新上传才能将云能力更新至 2.2.4 版本的云能力。

开通云开发

创建了第一个云开发小程序后，在使用云开发能力之前需要先开通云开发。在开发者工具工具栏左侧，单击“云开发”按钮即可开通云开发。云开发开通后自动获得一套云开发环境，各个环境相互隔离，每个环境都包含独立的数据库实例、文件存储空间、云函数配置等资源。每个环境都有唯一的环境 ID 标识，初始创建的环境自动成为默认环境。我们推荐，使用其中一个环境 `test` 作为测试环境，另一个环境 `release` ，作为正式环境。

获取环境 id

在【云开发控制台】 -> 【概览】中，如下图，复制【环境 ID】。如果你想先换或者新增环境，可以在右上角的按钮上操作。



小程序端配置环境 id

请到 client/app.js 文件中，填写上一步获取的【环境 ID】，如果不填，则表示使用默认环境。

```
wx.cloud.init({
env: '', // 前往云开发控制台获取环境id，如果使用默认环境则不需要填写
traceUser: true
});
```

云函数配置环境 id

请到 cloud/functions/addblog/index.js 文件中，填写【环境 ID】，如果不填，则表示使用默认环境。

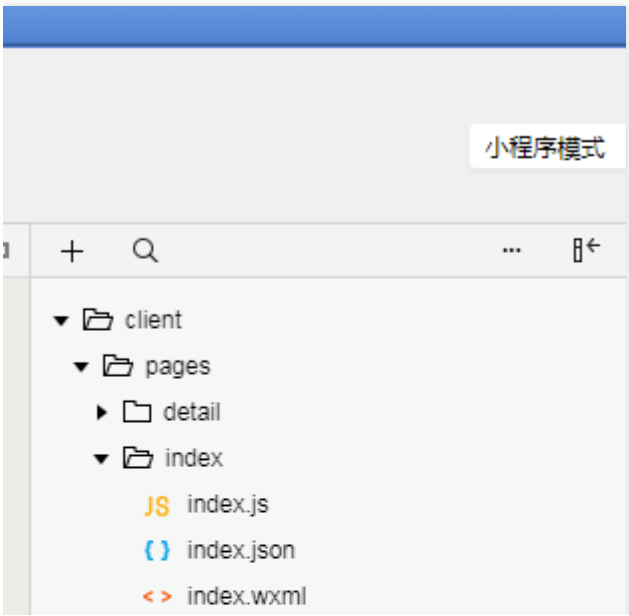
```
cloud.init({
env: '' // 前往云开发控制台获取环境id，如果使用默认环境则不需要填写
});
```

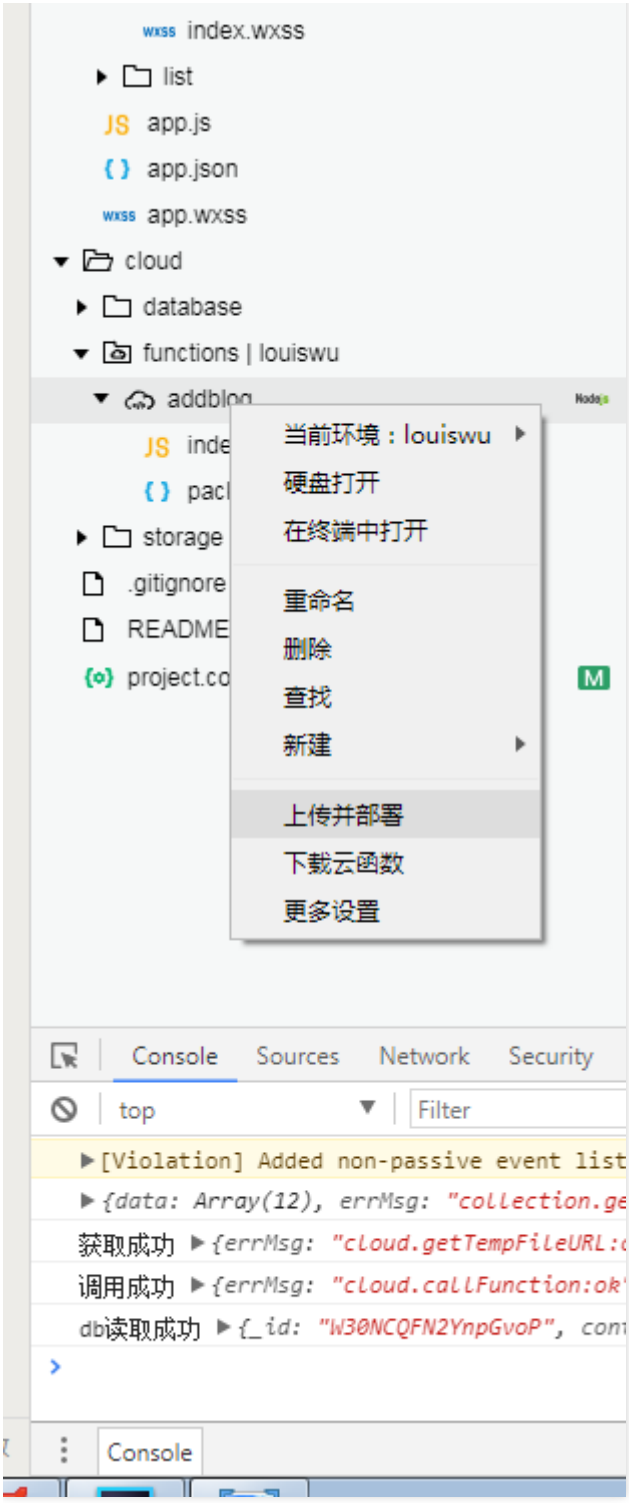
安装云函数依赖

```
// 安装依赖
cd cloud/functions/addblog/
npm install --production
```

上传云函数

在 IDE 中，右键云函数对应的文件夹，点击“上传并部署”菜单。





新建 collection

在小程序开发 IDE 中的，【云开发控制台】 > 【数据库】中，添加集合 `blog` 。

体验

单击小程序开发 IDE 中的“预览”，用微信扫一扫即可体验。恭喜你，你的第一个博客小程序已经做好啦！

云函数

概览

最近更新时间：2018-09-04 19:40:33

云函数是一段运行在云端的代码，无需管理服务器，在开发工具内编写、一键上传部署即可运行后端代码。

小程序内提供了专门用于云函数调用的 API。开发者可以在云函数内获取到每次调用的上下文（ `appid` 、 `openid` 等），无需维护复杂的鉴权机制，即可获取天然可信任的用户登录态（ `openid` ）。

比如我们如下定义一个云函数，命名为 add ，功能是将传入的两个参数 a 和 b 相加：

```
// index.js 是入口文件，云函数被调用时会执行该文件导出的 main 方法
// event 包含了调用端（小程序端）调用该函数时传过来的参数，同时还包含了用户登录态 `openId` 和小程序 `appId` 信息
exports.main = (event, context) => {
let { userInfo, a, b} = event
let { openId, appId} = userInfo // 这里获取到的 openId 和 appId 是可信的
let sum = a + b

return {
openId,
appId,
sum
}
}
```

在微信开发者工具中上传部署云函数后，我们在小程序中可以这么调用：

```
wx.cloud.callFunction({
// 需调用的云函数名
name: 'add',
// 传给云函数的参数
data: {
a: 12
b: 19,
},
// 成功回调
complete: console.log
})
// 当然 promise 方式也是支持的
wx.cloud.callFunction({
name: 'add',
```

```
  data: {
  a: 12,
  b: 19
  }
  }).then(console.log)
```

如需在云函数中操作数据库、管理云文件、调用其他云函数等操作，可使用官方提供的 npm 包 `wx-server-sdk` 进行操作。

更多的云函数开发指南，可参考以下章节：

- 写一个云函数
- 管理云函数
- 调用云函数
- 测试、日志与监控

写一个云函数

最近更新时间：2018-08-30 19:28:30

用 callback 返回数据

```
exports.main = (event, context, callback) => {
setTimeout(() => {
callback(null, {code: 0, data: 1});
}, 200);
}
```

用 return 返回数据

- 直接返回

```
exports.main = (event, context) => {

return {code: 0, data: 1};
}
```

- `async/await`

```
exports.main = async (event, context) => {

let result = new Promise((resolve, reject) => {
resolve({code: 0, data: 1});
});
return result;
}
```

管理云函数

最近更新时间：2018-08-30 19:28:36

配置云函数本地目录

在项目根目录中可以使用 `project.config.json` 文件，在其中定义 `cloudfunctionRoot` 字段，指定本地已存在的目录作为云函数的本地根目录。

云函数操作

在云函数根目录或者云函数目录上，通过鼠标右键，我们可以唤出右键菜单，完成以下操作

- 查看当前环境
- 切换环境
- 新建 `Node.js` 云函数
- 下载线上环境的云函数列表
- 下载线上环境的云函数代码并覆盖本地
- 对比本地代码和线上环境的代码
- 上传并部署云函数到线上环境

查看和切换环境

在云函数根目录上右键，在右键菜单中，可以查看当前对应的环境，同时可以切换环境，之后的所有右键菜单都是在这个环境下进行操作

新建 Node.js 云函数

在云函数根目录上右键，在右键菜单中，可以选择创建一个新的 `Node.js` 云函数，开发者工具在本地创建出以下目录和文件，同时在线上环境中创建出对应的云函数：

- 云函数目录：以云函数名字命名的目录，存放该云函数的所有代码
- `index.js` ：云函数入口文件，云函数被调用时实际执行的入口函数是 `index.js` 中导出的 `main` 方法
- `package.json` ：npm 包定义文件，其中默认定义了最新 `wx-server-sdk` 依赖

在创建成功后，工具会提示是否为该云函数立即安装本地依赖即 `wx-server-sdk` ，如是，则工具会开启终端执行 `npm install` 。

wx-server-sdk 是基于 tcb-admin-node 开发的 服务端 `SDK` ，与小程序端的接口使用方式一致，能获得更一体的开发体验。但如果想获得更高级的功能，可使用 `tcb-admin-node` 。

下载云函数列表

在云函数根目录上右键，在右键菜单中，我们可以将线上环境中的云函数列表同步到本地，开发者工具会根据云函数的名字，在本地中创建出对应的云函数目录

下载云函数

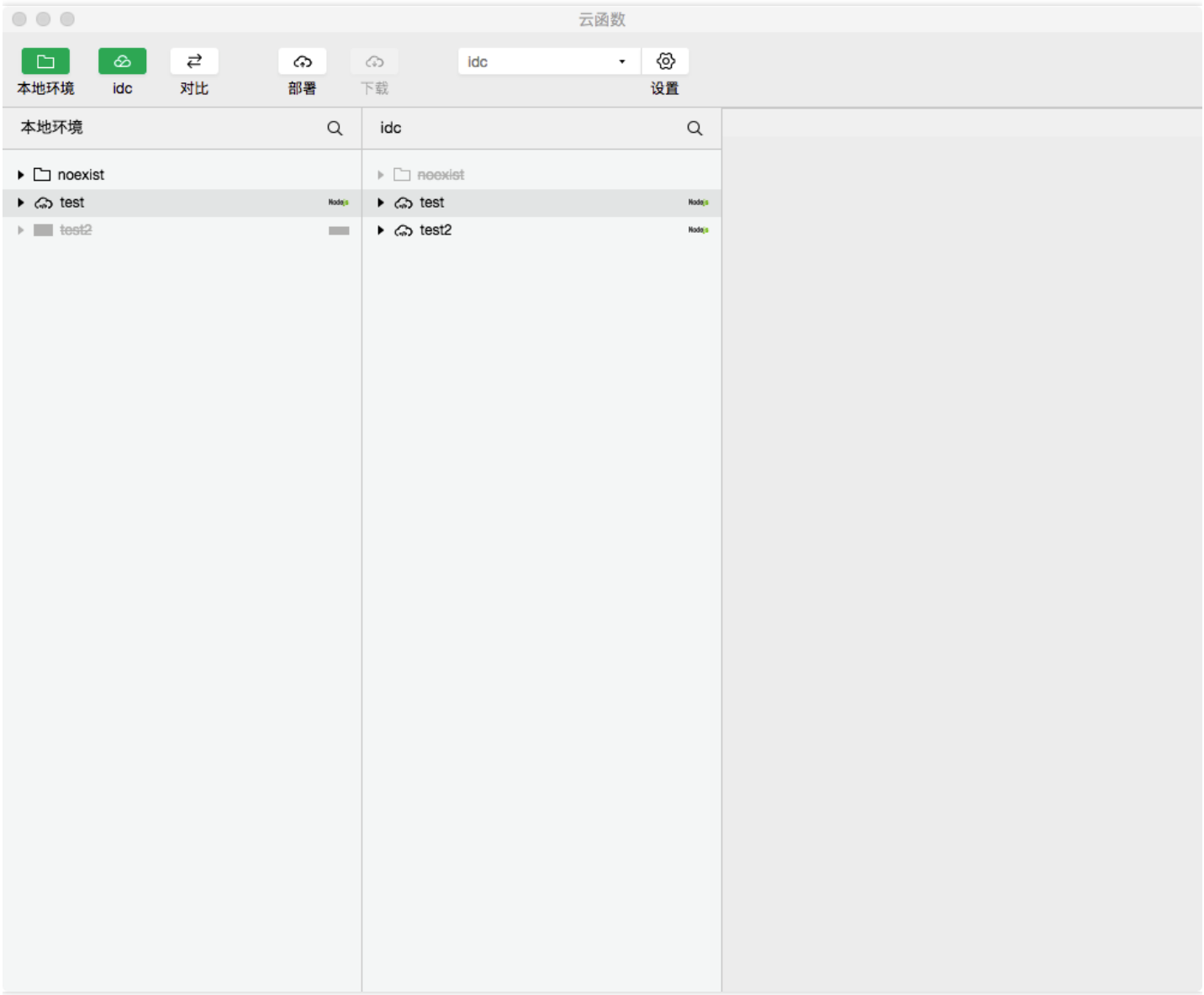
在一个云函数目录上右键可以在菜单中选择下载该云函数，云函数代码会被下载到指定目录。

上传并部署

在云函数目录上右键，在右键菜单中，我们可以将云函数整体打包上传并部署到线上环境中

更多设置

我们通过右键菜单的 “更多设置” 可以进入云函数的沉浸式交互场景，在这个场景里可以完成以上所有的云函数操作，在云目录上按 ctrl 可以进行多选批量操作


云函数
本地环境
idc
对比
部署
下载
idc
设置
本地环境
noexist
test
test2
idc
noexist
test
test2

调用云函数

最近更新时间：2018-08-30 19:28:43

小程序端

```
wx.cloud.callFunction({
// 云函数名称
name: 'add',
// 传给云函数的参数
data: {
a: 1,
b: 2,
},
})
.then(res => {
console.log(res.result) // 3
})
.catch(console.error);
```

服务端

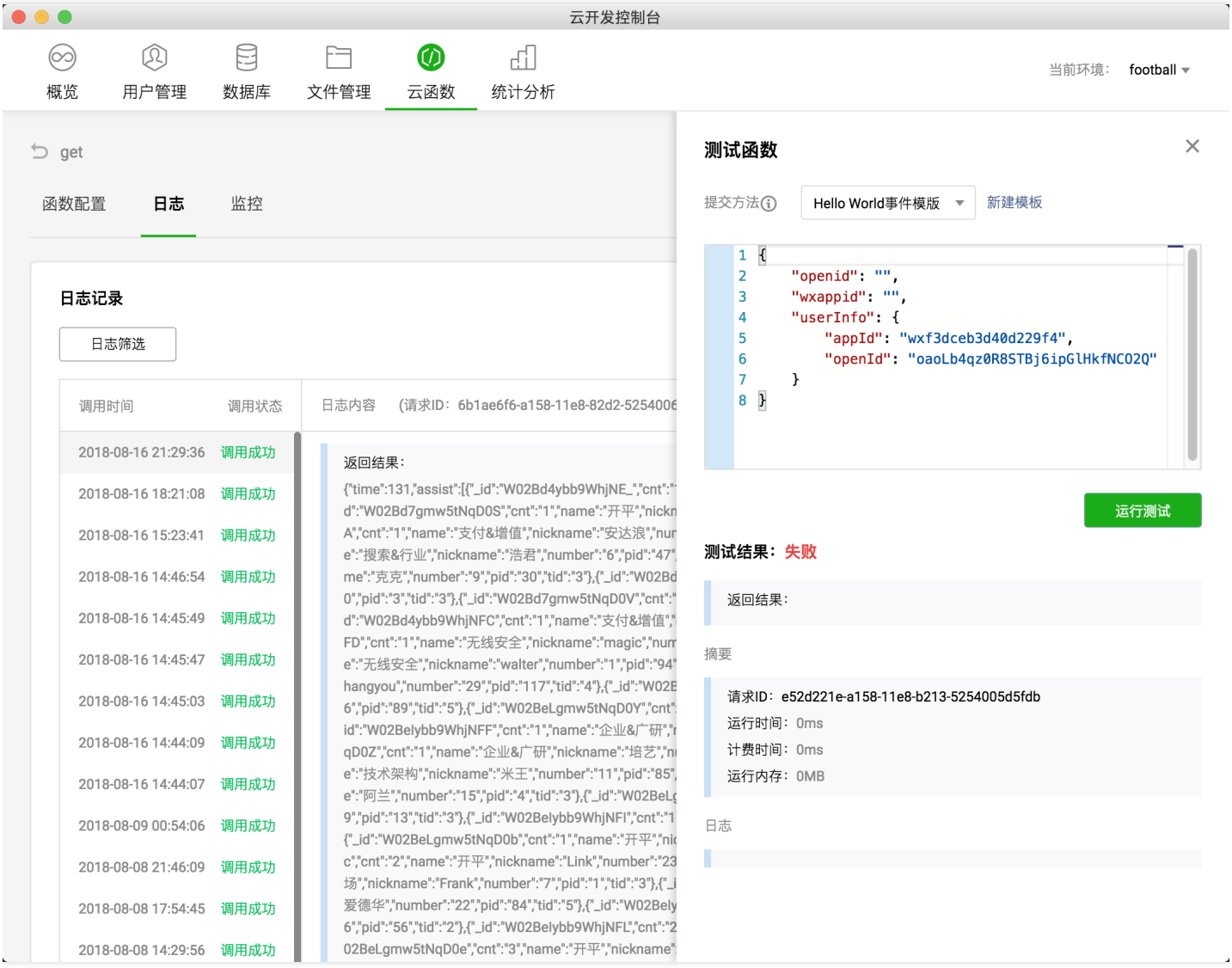
```
const app = require("tcb-admin-node");
app.init();

app.callFunction({
// 云函数名称
name: 'add',
// 传给云函数的参数
data: {
a: 1,
b: 2,
},
})
.then(res => {
console.log(res.result);
})
.catch(console.error)
```

测试、日志与监控

最近更新时间：2018-08-30 19:28:52

测试

云开发提供了云函数测试功能，可以方便地调试你的代码。在控制台的对应云函数的管理面板中，单击 “测试” 按钮即可打开测试弹窗。

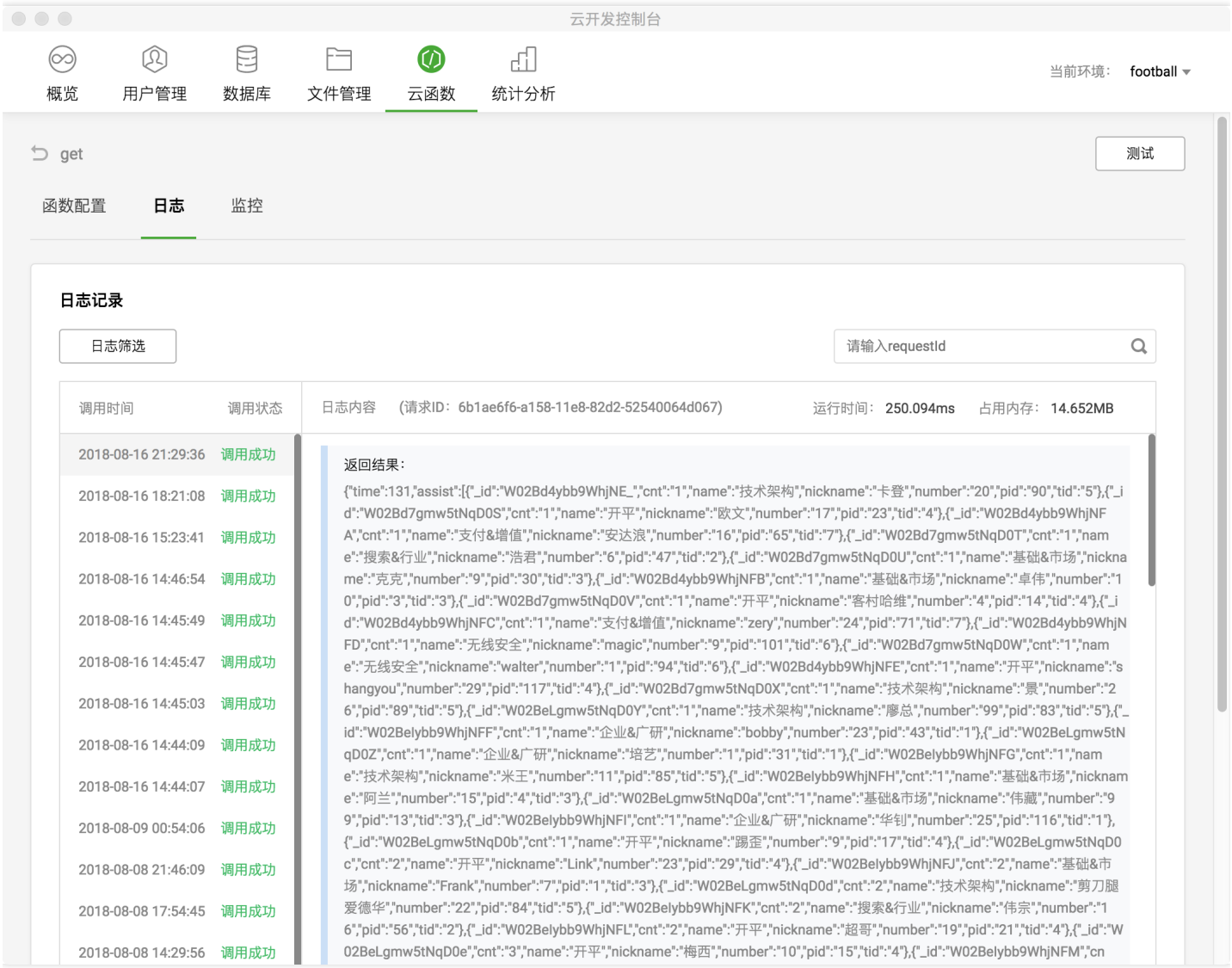


单击“提交方法”下拉菜单可以选择测试函数的模版方法，当前只支持 Hello World 事件模板。 模版在测试时作为 event 参数传递给函数。在“测试参数”的编辑器中输入想测试的参数后，点击“执行”按钮即可运行代码。执行完毕后将在“运行测试”栏显示运行结果。

除了可视化的云函数测试功能，我们还提供命令行工具 scf-cli, 助你在本地快速调试。

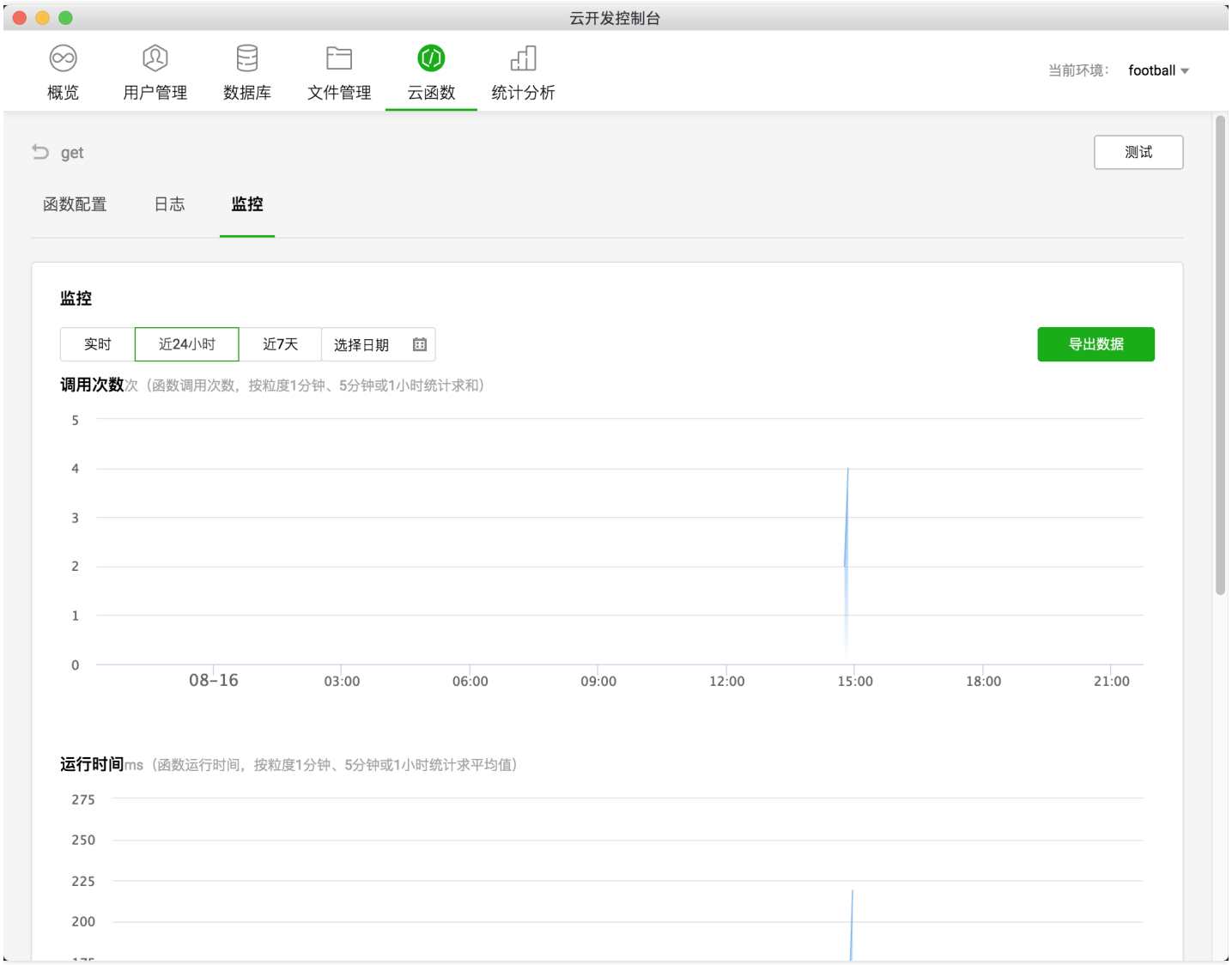
日志

在这里可以查看云函数的调用日志，方便开发者对开发调试。



监控

在这里可以查看云函数的调用次数、运行时间、错误次数。并支持将这些数据导出。



数据库
概览

最近更新时间：2018-09-04 19:41:10

TCB 提供了一个 NoSQL 数据库，数据库中的每条记录都是一个 JSON 格式的对象。一个数据库可以有多个集合（相当于关系型数据中的表），集合可看做一个 JSON 数组，数组中的每个对象就是一条记录（或称为文档），记录的格式是 JSON 对象。

关系型数据库和 JSON 数据库的概念对应关系如下表：

关系型	文档型
数据库 database	数据库 database
表 table	集合 collection
行 row	记录 record / doc
列 column	字段 field

以下是一个示例的集合数据，假设我们有一个 `books` 集合存放了图书记录，其中有两本书：

```
[
{
"_id": "Wzh76lk5_O_dt0vO",
"title": "The Catcher in the Rye",
"author": "J. D. Salinger",
"characters": [
"Holden Caulfield",
"Stradlater",
"Mr. Antolini"
],
"publishInfo": {
"year": 1951,
"country": "United States"
}
},
{
"_id": "Wzia0lk5_O_dt0vR",
"_openid": "ohl4L0Rnhq7vmmbT_DaNQa4ePaz0",
"title": "The Lady of the Camellias",
"author": "Alexandre Dumas fils",
```

```
"characters": [
"Marguerite Gautier",
"Armand Duval",
"Prudence",
"Count de Varville"
],
"publishInfo": {
"year": 1848,
"country": "France"
}
}
]
```

在图书信息中，我们用 `title`，`author` 来记录图书标题和作者，用 `characters` 数组来记录书中的主要人物，用 `publishInfo` 来记录图书的出版信息。在其中我们可以看到，字段既可以是字符串或数字，还可以是对象或数组，就是一个 JSON 对象。

每条记录都有一个 `_id` 字段用以唯一标志一条记录、一个 `_openid` 字段用以标志记录的创建者，即小程序的用户。需要特别注意的是，在管理端（云控制台和云函数）中创建的不会有 `_openid` 字段，因为这是属于管理员创建的记录。开发者可以自定义 `_id` ，但不可自定义和修改 `_openid` 。 `_openid` 是在文档创建时由系统根据小程序用户默认创建的，开发者可使用其来标识和定位文档。

数据库 API 分为小程序端和服务端两部分，小程序端 API 拥有严格的调用权限控制，开发者可在小程序内直接调用 API 进行非敏感数据的操作。对于有更高安全要求的数据，可在云函数内通过服务端 API 进行操作。云函数的环境是与客户端完全隔离的，在云函数上可以私密且安全的操作数据库。

数据库 API 包含增删改查的能力，使用 API 操作数据库只需三步：获取数据库引用、构造查询/更新条件、发出请求。以下是一个在小程序中查询数据库的发表于美国的图书记录的例子：

```
// 1. 获取数据库引用
const db = wx.cloud.database()
// 2. 构造查询语句
// collection 方法获取一个集合的引用
// where 方法传入一个对象，数据库返回集合中字段等于指定值的 JSON 文档。API 也支持高级的查询条件（比如大于、小于、in 等），具体见文档查看支持列表
// get 方法会触发网络请求，往数据库取数据
db.collection('books').where({
publishInfo: {
country: 'United States'
}
}).get({
success: function(res) {
// 输出 [{ "title": "The Catcher in the Rye", ... }]
console.log(res)
```

```
}
})
```

更多的数据库开发指南，可参考以下章节：

- 管理数据库
- 数据类型
- 初始化
- 插入数据
- 读取数据
- 构建查询条件
- 更新数据
- 删除数据
- 索引

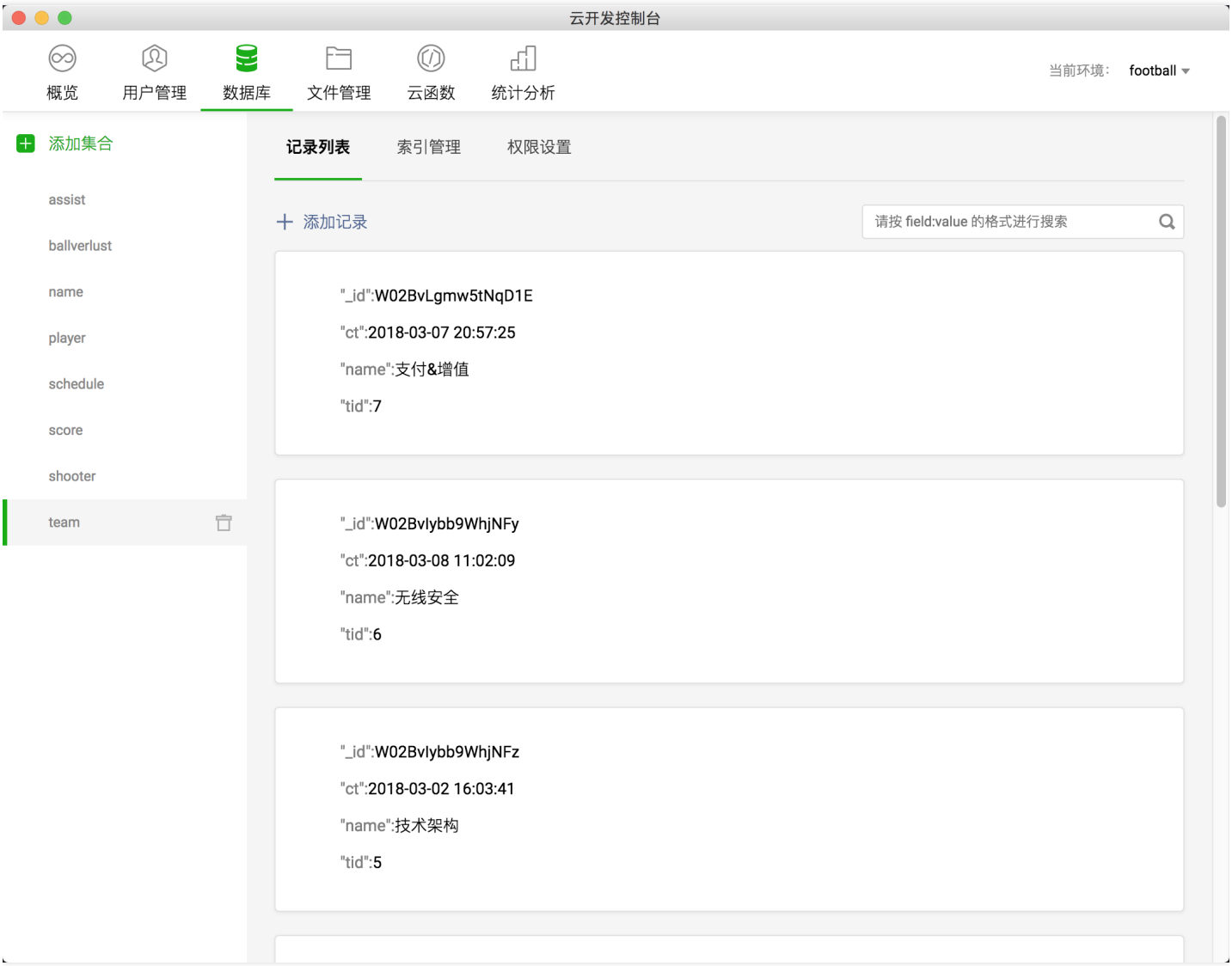
管理数据库

最近更新时间：2018-08-30 19:30:41

这一章我们将介绍如何在控制台中创建我们的第一个数据库集合、往集合上插入数据、以及在控制台中查看刚刚插入的数据。

创建第一个集合

打开控制台，选择 "数据库" 标签页，通过 "添加集合" 入口创建一个集合。假设我们要创建一个待办事项小程序，我们创建一个名为 todos 的集合。创建成功后，可以看到 todos 集合管理界面，界面中我们可以添加记录、查找记录、管理索引和管理权限。



创建第一条记录

控制台提供了可视化添加数据的交互界面，点击 "添加记录" 添加我们的第一条待办事项：

```
{
// 描述，String 类型
"description": "learn mini-program cloud service",
// 截止时间，Date 类型
"due": Date("2018-09-01"),
// 标签，Array 类型
"tags": [
"tech",
"mini-program",
"cloud"
],
// 个性化样式，Object 类型
"style": {
"color": "red"
},
// 是否已完成，Boolean 类型
"done": false
}
```

添加完成后可在控制台中查看到刚添加的数据。

权限控制

数据库的权限分为小程序端和管理端，管理端包括云函数端和控制台。小程序端运行在小程序中，读写数据库受权限控制限制，管理端运行在云函数上，拥有所有读写数据库的权限。云控制台的权限同管理端，拥有所有权限。小程序端操作数据库应有严格的安全规则限制。

初期我们对操作数据库开放以下几种权限配置，每个集合可以拥有一种权限配置，权限配置的规则是作用在集合的每个记录上的。出于易用性和安全性的考虑，云开发为云数据库做了小程序深度整合，在小程序中创建的每个数据库记录都会带有该记录创建者（即小程序用户）的信息，以 `_openid` 字段保存用户的 `openid` 在每个相应用户创建的记录中。因此，权限控制也相应围绕着一个用户是否应该拥有权限操作其他用户创建的数据展开。

以下按照权限级别从宽到紧排列如下：

- 仅创建者可写，所有人可读：数据只有创建者可写、所有人可读；比如文章。
- 仅创建者可读写：数据只有创建者可读写，其他用户不可读写；比如用私密相册。
- 仅管理端可写，所有人可读：该数据只有管理端可写，所有人可读；如商品信息。
- 仅管理端可读写：该数据只有管理端可读写；如后台用的不暴露的数据。

简而言之，管理端始终拥有读写所有数据的权限，小程序端始终不能写他人创建的数据，小程序端的记录的读写权限其实分为了“所有人可读，只有创建者可写“、”仅创建者可读写“、”所有人可读，仅管理端可写“、”所有人不可读，仅管理端可读写“。

对一个用户来说，不同模式在小程序端和管理端的权限表现如下：

端	小程序端	小程序端	小程序端	小程序端	管理端
权限	读自己创建的数据	写自己创建的数据	读他人创建的数据	写他人创建的数据	读写任意数据
仅创建者可写，所有人可读	√	√	√	×	√
仅创建者可读写	√	√	×	×	√
仅管理端可写，所有人可读	√	×	√	×	√
仅管理端可读写：该数据只有管理端可读写	×	×	×	×	√

数据类型

最近更新时间：2018-08-30 19:30:47

云开发数据库提供以下几种数据类型：

- String：字符串
- Number：数字
- Object：对象
- Array：数组
- Bool：布尔值
- GeoPoint：地理位置点
- Date：时间
- Null

下面对几个需要额外说明的字段做下补充说明。

Date

`Date` 类型用于表示时间，精确到毫秒，在小程序端可用 `JavaScript` 内置 `Date` 对象创建。需要特别注意的是，在小程序端创建的时间是客户端时间，不是服务端时间，这意味着在小程序端的时间与服务端时间不一定吻合，如果需要使用服务端时间，应该用 `API` 中提供的 `serverDate` 对象来创建一个服务端当前时间的标记，当使用了 `serverDate` 对象的请求抵达服务端处理时，该字段会被转换成服务端当前的时间，更棒的是，我们在构造 `serverDate` 对象时还可通过传入一个有 `offset` 字段的对象来标记一个与当前服务端时间偏移 offset 毫秒的时间，这样我们就可以达到比如如下效果：指定一个字段为服务端时间往后一个小时。

那么当我们需要使用客户端时间时，存放 `Date` 对象和存放毫秒数是否是一样的效果呢？不是的，我们的数据库有针对日期类型的优化，建议大家使用时都用 `Date` 或 serverDate 构造时间对象。

GeoPoint

`GeoPoint` 类型用于表示地理位置点，用经纬度唯一标记一个点，这是一个特殊的数据存储类型。注意，如果需要对类型为地理位置的字段进行查找，一定要建立地理位置索引。

具体的地理位置 `API` 可参考 Geo API 文档。

初始化

最近更新时间：2018-08-30 19:30:55

小程序端

在开始使用数据库 API 进行增删改查操作之前，需要先获取数据库的引用。以下调用获取默认环境的数据库的引用：

```
const db = wx.cloud.database();
```

如需获取其他环境的数据库引用，可以在调用时传入一个含 `config` 字段的参数，在其中通过 `env` 字段指定要使用的环境。此时方法会返回一个对测试环境数据库的引用。

示例：假设有一个环境名为 `test` ，用做测试环境，那么可以如下获取测试环境数据库：

```
const testDB = wx.cloud.database({
config: {
env: 'test'
}
});
```

要操作一个集合，需先获取它的引用。在获取了数据库的引用后，就可以通过数据库引用上的 `collection` 方法获取一个集合的引用了，比如获取待办事项清单集合：

```
const todos = db.collection('todos');
```

获取集合的引用并不会发起网络请求取拉取它的数据，我们可以通过此引用在该集合上进行增删查改的操作，除此之外，还可以通过集合上的 `doc` 方法来获取集合中一个指定 `ID` 的记录的引用。同理，记录的引用可以用于对特定记录进行更新和删除操作。

假设我们有一个待办事项的 `ID` 为 `todo-identifiant-aleatoire` ，那么我们可以通过 `doc` 方法获取它的引用：

```
const todo = db.collection('todos').doc('todo-identifiant-aleatoire');
```

服务端

```
const app = require("tcb-admin-node");

// 初始化资源
// 云函数下不需要secretId和secretKey。
// env如果不指定将使用默认环境
app.init({
```

```
secretId: 'xxxxx',
secretKey: 'xxxx',
env: 'xxx'
});

//云函数下使用默认环境
app.init();

//云函数下指定环境
app.init({
env: 'xxx'
});
```

插入数据

最近更新时间：2018-08-30 19:31:01

可以通过在集合对象上调用 add 方法往集合中插入一条记录。还是用待办事项清单的例子，比如我们想新增一个待办事项：

小程序端

```
const db = wx.cloud.database();
db.collection('todos').add({
// data 字段表示需新增的 JSON 数据
data: {
// _id: 'todo-identifiant-aleatoire', // 可选自定义 _id，在此处场景下用数据库自动分配的就可以了
description: "learn cloud database",
due: new Date("2018-09-01"),
tags: [
"cloud",
"database"
],
// 为待办事项添加一个地理位置
location: new db.Geo.Point(23, 113),
done: false
},
})
.then(res => {
console.log(res)
})
.catch(console.error);
```

服务端

```
const app = require('tcb-admin-node');
app.init();
const db = app.database();

db.collection('todos').add({
// data 字段表示需新增的 JSON 数据
data: {
// _id: 'todo-identifiant-aleatoire', // 可选自定义 _id，在此处场景下用数据库自动分配的就可以了
description: "learn cloud database",
due: new Date("2018-09-01"),
tags: [
```

```
  "cloud",
  "database"
  ],
  // 为待办事项添加一个地理位置
  location: new db.Geo.Point(23, 113),
  done: false
  },
  })
  .then(res => {
  console.log(res)
  })
  .catch(console.error);
```

在创建成功之后，我们可以在控制台中查看到刚新增的数据。

读取数据

最近更新时间：2018-08-30 19:31:36

在记录和集合上都有提供 get 方法用于获取单个记录或集合中多个记录的数据。

假设我们已有一个集合 todos，其中包含以下格式记录：

```
[
{
_id: 'todo-identifiant-aleatoire',
_openid: 'user-open-id', // 假设用户的 openid 为 user-open-id
description: "learn cloud database",
due: Date("2018-09-01"),
progress: 20,
tags: [
"cloud",
"database"
],
style: {
color: 'white',
size: 'large'
},
location: Point(23.33, 113.33),
done: false
},
{
_id: 'todo-identifiant-aleatoire-2',
_openid: 'user-open-id', // 假设用户的 openid 为 user-open-id
description: "write a novel",
due: Date("2018-12-25"),
progress: 50,
tags: [
"writing"
],
style: {
color: 'yellow',
size: 'normal'
},
location: Point(23.22, 113.22),
done: false
}
// more...
]
```

小程序端

获取一个记录的数据

我们先来看看如何获取一个记录的数据，假设我们已有一个 ID 为 todo-identifiant-aleatoire 的在集合 todos 上的记录，那么我们可以通过在该记录的引用调用 get 方法获取这个待办事项的数据：

```
db.collection('todos').doc('todo-identifiant-aleatoire').get().then((res) => {
// res.data 包含该记录的数据
console.log(res.data);
});
```

获取多个记录的数据

我们也可以一次性获取多条记录。通过调用集合上的 where 方法可以指定查询条件，再调用 get 方法即可只返回满足指定查询条件的记录，比如获取用户的所有未完成的待办事项：

```
db.collection('todos').where({
_openid: 'user-open-id',
done: false
})
.get().then((res) => {
// res.data 是包含以上定义的两条记录的数组
console.log(res.data);
});
```

where 方法接收一个对象参数，该对象中每个字段和它的值构成一个需满足的匹配条件，各个字段间的关系是 "与" 的关系，即需同时满足这些匹配条件，在这个例子中，就是查询出 todos 集合中 _openid 等于 user-open-id 且 done 等于 false 的记录。在查询条件中我们也可以指定匹配一个嵌套字段的值，比如找出自己的标为黄色的待办事项：

```
db.collection('todos').where({
_openid: 'user-open-id',
style: {
color: 'yellow'
}
})
.get().then((res) => {
console.log(res.data);
});
```

也可以用 "点表示法" 表示嵌套字段：

```
db.collection('todos').where({
_openid: 'user-open-id',
'style.color': 'yellow'
})
.get().then((res) => {
console.log(res.data);
});
```

获取一个集合的数据

如果要获取一个集合的数据，比如获取 `todos` 集合上的所有记录，可以在集合上调用 `get` 方法获取，但通常不建议这么使用，在小程序中我们需要尽量避免一次性获取过量的数据，只应获取必要的数据。为了防止误操作以及保护小程序体验，在获取集合数据时服务器一次最多返回 `20` 条记录。开发者可以通过 `limit` 方法指定需要获取的记录数量，但不能超过 `20` 条。

```
db.collection('todos').get().then((res) => {
// res.data 是一个包含集合中有权限访问的所有记录的数据，不超过 20 条
console.log(res.data);
});
```

服务端

服务端的使用与小程序端的大同小异，下面只是简单举些例子：

```
const app = require('tcb-admin-node');
app.init();
const db = app.database();

// 获取一个记录的数据
db.collection('todos').doc('todo-identifiant-aleatoire').get().then((res) => {
// res.data 包含该记录的数据
console.log(res.data);
});

// 获取多个记录的数据
db.collection('todos').where({
_openid: 'user-open-id',
done: false
})
```

```
.get().then((res) => {
// res.data 是包含以上定义的两条记录的数组
console.log(res.data);
});

// 获取一个集合的数据
db.collection('todos').get().then((res) => {
// res.data 是一个包含集合中有权限访问的所有记录的数据，不超过 20 条
console.log(res.data);
});
```

构建查询条件

最近更新时间：2018-08-30 19:31:52

使用数据库 API 提供的 where 方法我们可以构造复杂的查询条件完成复杂的查询任务。在本节中我们还是使用上一章 节中使用的示例数据。以下这些构建查询条件，小程序端与服务端 SDK 基本都保持一致。

查询指令

假设我们需要查询进度大于 30% 的待办事项，那么传入对象表示全等匹配的方式就无法满足了，这时就需要用到查询指令。数据库 API 提供了大于、小于等多种查询指令，这些指令都暴露在 db.command 对象上。比如查询进度大于 30% 的待办事项：

```
const _ = db.command
db.collection('todos').where({
// gt 方法用于指定一个 "大于" 条件，此处 _.gt(30) 是一个 "大于 30" 的条件
progress: _.gt(30)
})
.get()
.then((res) => {
console.log(res.data);
});
```

API 提供了以下查询指令：

查询指令	说明
eq	等于
neq	不等于
lt	小于
lte	小于或等于
gt	大于
gte	大于或等于
in	字段值在给定数组中
nin	字段值不在给定数组中

具体的查询指令 API 文档可参考 数据库文档。

逻辑指令

除了指定一个字段满足一个条件之外，我们还可以通过指定一个字段需同时满足多个条件，比如用 `and` 逻辑指令查询进度在 30% 和 70% 之间的待办事项：

```
const _ = db.command
db.collection('todos').where({
// and 方法用于指定一个 "与" 条件，此处表示需同时满足 _.gt(30) 和 _.lt(70) 两个条件
progress: _.gt(30).and(_.lt(70))
})
.get()
.then((res) => {
console.log(res.data);
});
```

既然有 `and` ，当然也有 `or` 了，比如查询进度为 0 或 100 的待办事项：

```
const _ = db.command
db.collection('todos').where({
// or 方法用于指定一个 "或" 条件，此处表示需满足 _.eq(0) 或 _.eq(100)
progress: _.eq(0).or(_.eq(100))
})
.get().then((res) => {
console.log(res.data);
});
```

如果我们需要跨字段进行 "或" 操作，可以做到吗？答案是肯定的， `or` 指令还可以用来接受多个（可以多于两个）查询条件，表示需满足多个查询条件中的任意一个，比如我们查询进度小于或等于 50% 或颜色为白色或黄色的待办事项：

```
const _ = db.command
db.collection('todos').where(_.or([
{
progress: _.lte(50)
},
{
style: {
color: _.in(['white', 'yellow'])
}
}
]))
.get()
.then((res) => {
```

```
  console.log(res.data);
});
```

具体的逻辑指令 API 文档可参考 数据库文档。

更新数据

最近更新时间：2018-08-30 19:32:11

在这章节我们一起看看如何使用数据库 API 完成数据更新，在本节中我们还是沿用 读取数据 章节中使用的数据。

更新数据主要有两个方法：

API	说明
update	局部更新一个或多个记录
set	替换更新一个记录

小程序端

局部更新

使用 `update` 方法可以局部更新一个记录或一个集合中的记录，局部更新意味着只有指定的字段会得到更新，其他字段不受影响。

比如我们可以用以下代码将一个待办事项置为已完成：

```
db.collection('todos').doc('todo-identifiant-aleatoire').update({
// data 传入需要局部更新的数据
data: {
// 表示将 done 字段置为 true
done: true
}
})
.then((res) => {
console.log(res.data);
});
```

除了用指定值更新字段外，数据库 API 还提供了一系列的更新指令用于执行更复杂的更新操作，更新指令可以通过 `db.command` 取得：

更新指令	说明
set	设置字段为指定值
remove	删除字段
inc	原子自增字段值

更新指令	说明
mul	原子自乘字段值
push	如字段值为数组，往数组尾部增加指定值
pop	如字段值为数组，从数组尾部删除一个元素
shift	如字段值为数组，从数组头部删除一个元素
unshift	如字段值为数组，往数组头部增加指定值

比如我们可以将一个待办事项的进度 +10%：

```
const _ = db.command
db.collection('todos').doc('todo-identifiant-aleatoire').update({
data: {
// 表示指示数据库将字段自增 10
progress: _.inc(10)
}
})
.then((res) => {
console.log(res.data);
});
```

用 `inc` 指令而不是取出值、加 10 再写进去的好处在于这个写操作是个原子操作，不会受到并发写的影响，比如同时有两名用户 A 和 B 取了同一个字段值，然后分别加上 10 和 20 再写进数据库，那么这个字段最终结果会是加了 20 而不是 30。如果使用 `inc` 指令则不会有这个问题。

如果字段是个数组，那么我们可以使用 `push`、`pop`、`shift` 和 `unshift` 对数组进行原子更新操作，比如给一条待办事项加多一个标签：

```
const _ = db.command
db.collection('todos').doc('todo-identifiant-aleatoire').update({
data: {
tags: _.push('mini-program')
}
})
.then((res) => {
console.log(res.data);
});
```

可能读者已经注意到我们提供了 `set` 指令，这个指令有什么用呢？这个指令的用处在于更新一个字段值为另一个对象。比如如下语句是更新 `style.color` 字段为 'blue' 而不是把 `style` 字段更新为 `{ color: 'blue' }` 对象：

```
const _ = db.command
db.collection('todos').doc('todo-identifiant-aleatoire').update({
data: {
style: {
color: 'blue'
}
}
})
.then((res) => {
console.log(res.data);
});
```

如果需要将这个 `style` 字段更新为另一个对象，可以使用 `set` 指令：

```
const _ = db.command
db.collection('todos').doc('todo-identifiant-aleatoire').update({
data: {
style: _.set({
color: 'blue'
})
}
})
.then((res) => {
console.log(res.data);
});
```

更完整详细的更新指令可以参考 数据库文档

替换更新

如果需要替换更新一条记录，可以在记录上使用 `set` 方法，替换更新意味着用传入的对象替换指定的记录：

```
const _ = db.command
db.collection('todos').doc('todo-identifiant-aleatoire').set({
data: {
description: "learn cloud database",
due: new Date("2018-09-01"),
tags: [
"cloud",
"database"
],
style: {
color: "skyblue"
},
location: new db.Geo.Point(23, 113),
```

```
done: false
}
})
.then((res) => {
console.log(res.data);
});
```

如果指定 ID 的记录不存在，则会自动创建该记录，该记录将拥有指定的 ID。

服务端

更新多个数据

如果需要更新多个数据，需在 Server 端进行操作（云函数），在 `where` 语句后同样的调用 `update` 方法即可，比如将所有未完待办事项的进度加 10%：

```
// 使用了 async await 语法
const app = require('tcb-admin-node');
app.init();
const db = app.database();
const _ = db.command;

exports.main = async (event, context) => {
try {
return await db.collection('todos').where({
done: false
})
.update({
data: {
progress: _.inc(10)
},
});
} catch(e) {
console.error(e);
}
}
```

删除数据

最近更新时间：2018-08-30 19:32:27

在这章节我们一起看看如何使用数据库 API 完成数据删除，在本节中我们还是沿用 读取数据 章节中使用的数据。

小程序端

删除一条记录

对记录使用 `remove` 方法可以删除该条记录，比如：

```
db.collection('todos').doc('todo-identifiant-aleatoire').remove({
success: function(res) {
console.log(res.data)
}
})
```

服务端

删除多条记录

如果需要更新多个数据，需在 Server 端进行操作（云函数）。可通过 `where` 语句选取多条记录执行删除，只有有权限删除的记录会被删除。比如删除所有已完成的待办事项：

```
// 使用了 async await 语法
const app = require('tcb-admin-node');
app.init();
const db = app.database();
const _ = db.command;

exports.main = async (event, context) => {
try {
return await db.collection('todos').where({
done: true
}).remove();
} catch(e) {
console.error(e);
}
}
```

```

在大多数情况下，我们希望用户只能操作自己的数据（自己的代表事项），不能操作其他人的数据（其他人的待办事项），这就需要引入权限控制了。
```

索引

最近更新时间：2018-08-30 19:32:34

索引管理

建立索引是保证数据库性能、保证小程序体验的重要手段。我们应为所有需要成为查询条件的字段建立索引。建立索引的入口在控制台中，可分别对各个集合的字段添加索引。

单字段索引

对需要作为查询条件筛选的字段，我们可以创建单字段索引。如果需要对嵌套字段进行索引，那么可以通过 "点表示法" 用点连接起嵌套字段的名称。比如我们需要对如下格式的记录中的 color 字段进行索引时，可以用 style.color 表示。

```
{
_id: '',
style: {
color: ''
}
}
```

在设置单字段索引时，指定排序为升序或降序并没有关系。在需要对索引字段按排序查询时，数据库能够正确的对字段排序，无论索引设置为升序还是降序。

组合索引

组合索引即一个索引包含多个字段。当查询条件使用的字段包含在索引定义的所有字段或前缀字段里时，会命中索引，优化查询性能。索引前缀即组合索引的字段中定义的前 1 到多个字段，如有在 `A`，`B`，`C` 三个字段定义的组合索引 `A`，`B`，`C` ，那么 `A` 和 `A`，`B` 都属于该索引的前缀。

组合索引具有以下特点：

1. 字段顺序决定索引效果

定义组合索引时，多个字段间的顺序不同是会有不同的索引效果的。比如对两个字段 `A` 和 `B` 进行索引，定义组合索引为 `A`，`B` 与定义组合索引为 `B`，`A` 是不同的。当定义组合索引为 `A`，`B` 时，索引会先按 `A` 字段排序再按 `B` 字段排序。因此当组合索引设为 `A`，`B` 时，即使我们没有单独对字段 `A` 设立索引，但对字段 `A` 的查询可以命中 `A`，`B` 索引。需要注意的是，此时对字段 `B` 的查询是无法命中 `A`，`B` 索引的，因为 `B` 不属于索引 `A`，`B` 的前缀之一。

1. 字段排序决定排序查询是否可以命中索引

加入我们对字段 A 和 B 设置以下索引：

A: 升序
B: 降序

那么当我们查询需要对 A，B 进行排序时，可以指定排序结果为 A 升序 B 降序或 A 降序 B 升序，但不能指定为 A 升序 B 升序或 A 降序 B 降序。

文件存储

概览

最近更新时间：2018-09-11 11:57:54

TCB 为开发者提供了存储空间，包含了上传文件到云端、带权限管理的云端下载能力，开发者可以在小程序端和云函数端通过 API 使用云文件存储功能。

在小程序端可以分别调用 `wx.cloud.uploadFile` 和 `wx.cloud.downloadFile` 完成上传和下载云文件操作。下面简单的几行代码，即可实现在小程序内让用户选择一张图片，然后上传到云端管理的功能：

```
// 让用户选择一张图片
wx.chooseImage({
success: chooseResult => {
// 将图片上传至云存储空间
wx.cloud.uploadFile({
// 指定上传到的云路径
cloudPath: 'my-photo.png',
// 指定要上传的文件的小程序临时文件路径
filePath: chooseResult.tempFilePaths[0],
// 成功回调
success: res => {
console.log('上传成功', res)
},
})
},
})
```

上传完成后可在云控制台中看到刚上传的图片。

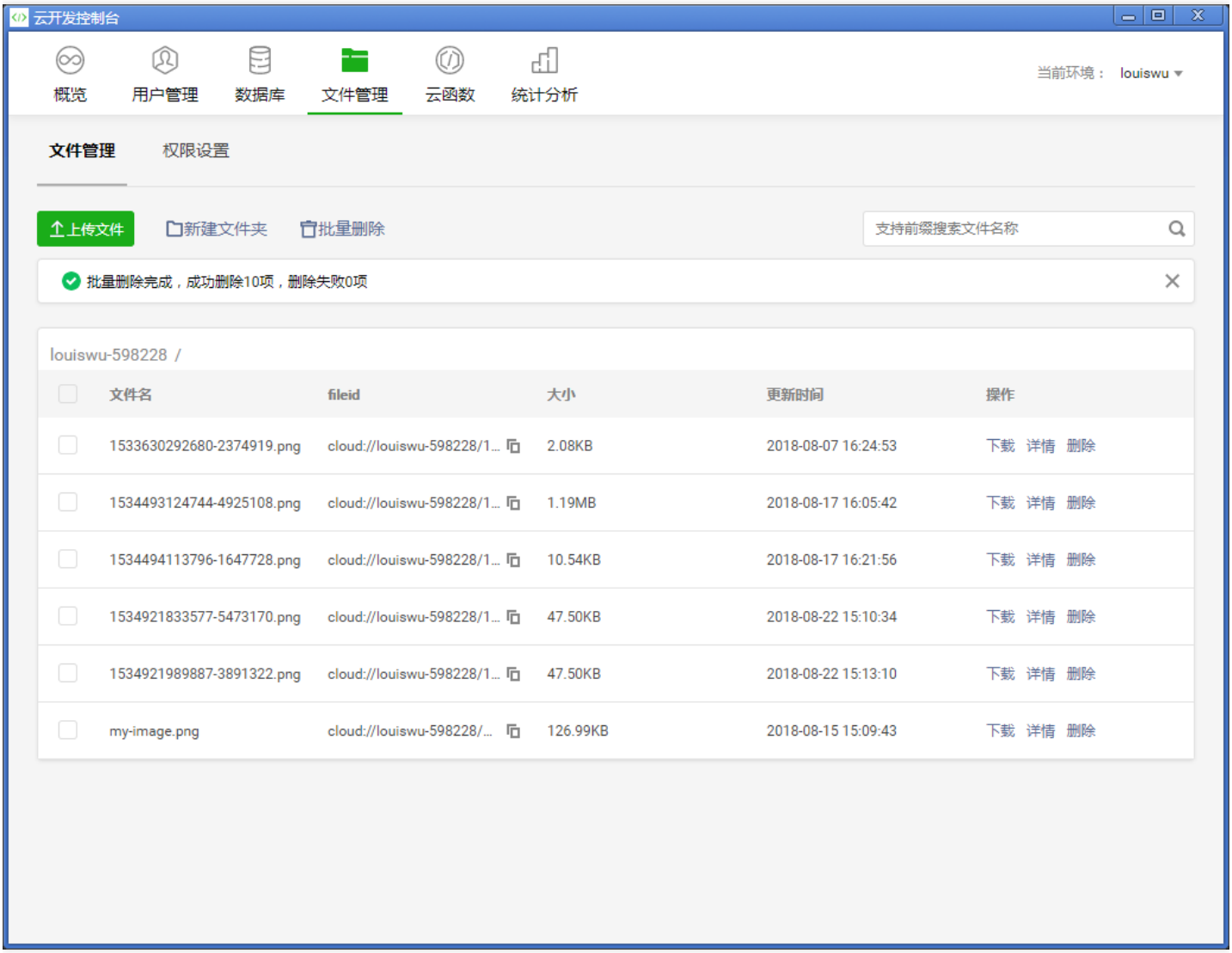
更多的存储开发指南，可参考以下章节：

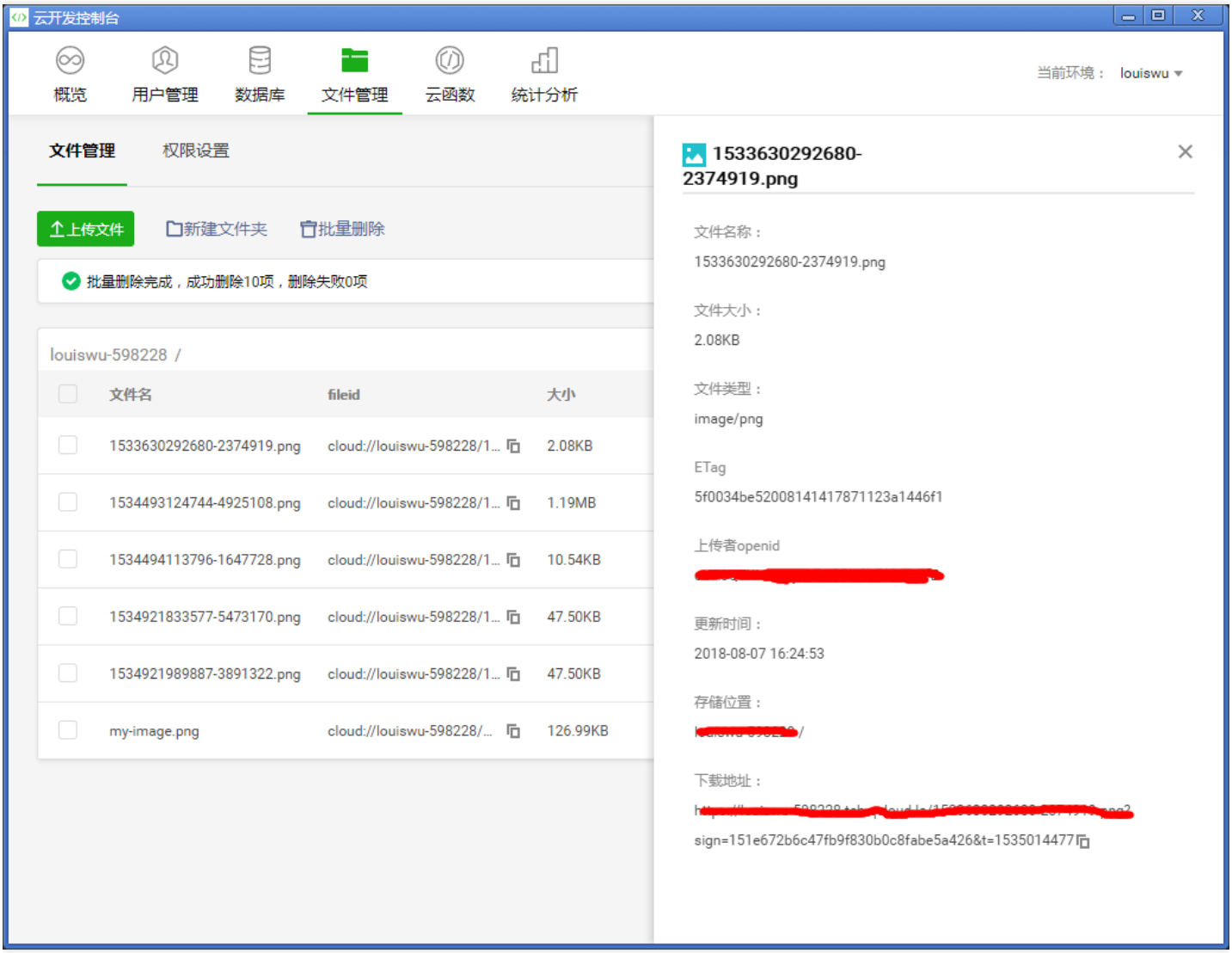
- 管理文件
- 上传文件
- 下载文件
- 删除文件
- 换取临时链接
- 组件支持

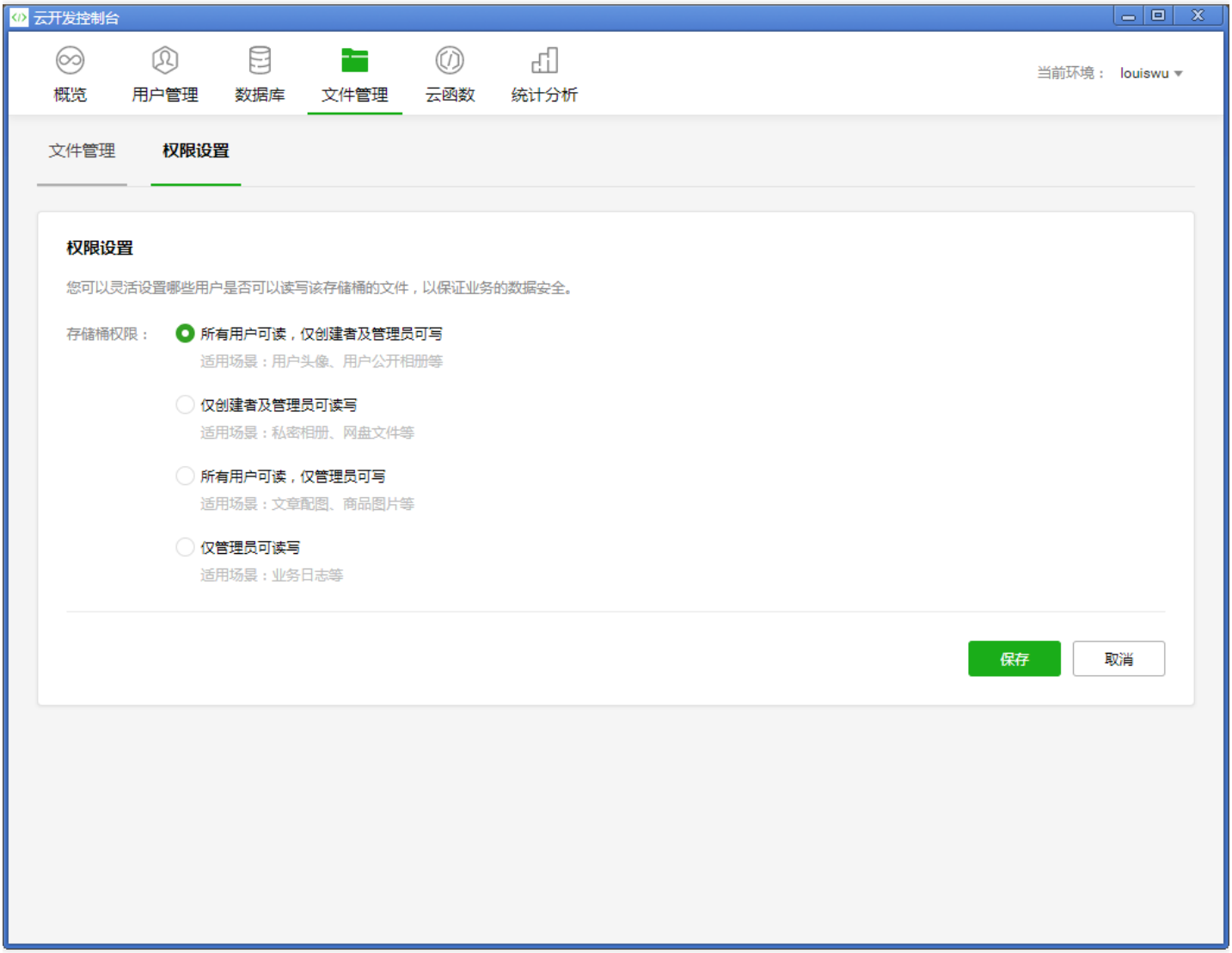
管理文件

最近更新时间：2018-08-30 19:33:04

在控制台中，选择文件管理标签页，可以在此看到云存储空间中所有的文件，还可以查看文件的详细信息、控制存储空间的读写权限。




云开发控制台
概览
用户管理
数据库
文件管理
云函数
统计分析
当前环境： louiswu
文件管理
权限设置
上传文件
新建文件夹
批量删除
批量删除完成，成功删除10项，删除失败0项
louiswu-598228 /
文件名
fileid
大小
1533630292680-2374919.png
cloud://louiswu-598228/1...
2.08KB
1534493124744-4925108.png
cloud://louiswu-598228/1...
1.19MB
1534494113796-1647728.png
cloud://louiswu-598228/1...
10.54KB
1534921833577-5473170.png
cloud://louiswu-598228/1...
47.50KB
1534921989887-3891322.png
cloud://louiswu-598228/1...
47.50KB
my-image.png
cloud://louiswu-598228/...
126.99KB
1533630292680-2374919.png
文件名称：
1533630292680-2374919.png
文件大小：
2.08KB
文件类型：
image/png
ETag
5f0034be52008141417871123a1446f1
上传者openid
更新时间：
2018-08-07 16:24:53
存储位置：
下载地址：
sign=151e672b6c47fb9f830b0c8fabe5a426&t=1535014477


云开发控制台
概览
用户管理
数据库
文件管理
云函数
统计分析
当前环境： louiswu
文件管理
权限设置
权限设置
您可以灵活设置哪些用户是否可以读写该存储桶的文件，以保证业务的数据安全。
存储桶权限：
所有用户可读，仅创建者及管理员可写
适用场景：用户头像、用户公开相册等
仅创建者及管理员可读写
适用场景：私密相册、网盘文件等
所有用户可读，仅管理员可写
适用场景：文章配图、商品图片等
仅管理员可读写
适用场景：业务日志等
保存
取消

上传文件

最近更新时间：2018-08-30 19:33:10

小程序端

```
wx.cloud.uploadFile({
cloudPath: 'example.png', // 上传至云端的路径
filePath: '', // 小程序临时文件路径
}).then((res) => {
// 返回文件 ID
console.log(res.fileID)
}).catch(console.err);
```

服务端

```
const app = require("tcb-admin-node");
const fs = require("fs");
app.init();

app.uploadFile({
cloudPath: "test-admin.jpeg",
fileContent: fs.createReadStream(`${__dirname}/cos.jpeg`)
}).then((res) => {
// 返回文件 ID
console.log(res.fileID);
}).catch(console.error);
```

下载文件

最近更新时间：2018-08-30 19:33:16

小程序端

```
wx.cloud.downloadFile({
fileID: '', // 文件 ID
}).then((res) => {
// 返回临时文件路径
console.log(res.tempFilePath)
}).catch(console.error);
```

服务端

```
const app = require("tcb-admin-node");
app.init();
app.downloadFile({
fileID: "cloud://aa-99j9f/my-photo.png",
// tempFilePath: '/tmp/test/storage/my-photo.png'
}).then((res) => {
// 返回临时文件路径
console.log(res.tempFilePath)
}).catch(console.error);
```

删除文件

最近更新时间：2018-08-30 19:33:23

小程序端

```
wx.cloud.deleteFile({
fileList: ['a7xzcb'], // 文件 ID 数组
success: res => {
// handle success
console.log(res.fileList)
},
fail: console.error
}).then((res) => {
console.log(res.fileList)
}).catch(console.error);
```

服务端

```
const app = require('tcb-admin-node');
app.init();

app.deleteFile({
fileList: ['a7xzcb+yilTJE4NPSQQW5EYks']
}).then((res) => {
// handle success
console.log(res.fileList)
}).catch(console.error);
```

换取临时链接

最近更新时间：2018-08-30 19:33:31

小程序端

```
wx.cloud.getTempFileURL({
fileList: ['cloud://xxx.png']
}).then((res) => {
// fileList 是一个有如下结构的对象数组
// [{
// fileID: 'cloud://xxx.png', // 文件 ID
// tempFileURL: '', // 临时文件网络链接
// maxAge: 120 * 60 * 1000, // 有效期
// }]
console.log(res.fileList)
}).catch(console.error);
```

服务端

```
const app = require('tcb-admin-node');
app.init();

app.getTempFileURL({
fileList: ['cloud://xxx.png']
}).then((res) => {
// fileList 是一个有如下结构的对象数组
// [{
// fileID: 'cloud://xxx.png', // 文件 ID
// tempFileURL: '', // 临时文件网络链接
// maxAge: 120 * 60 * 1000, // 有效期
// }]
console.log(res.fileList)
}).catch(console.error);
```

组件支持

最近更新时间：2018-09-19 11:59:51

部份小程序的组件，支持传入云文件的 `file ID` ，具体支持的情况，请访问小程序的 组件支持 文档。