第3章

云数据库

云数据库提供高性能的数据库写入和查询服务。通过腾讯云开发(Tencent Cloud Base, TCB)的SDK,可以直接在客户端对数据进行读写,也可以在云函数中读写数据,还可以通过控制台对数据进行可视化的增、删、查、改等操作。微信小程序云开发所使用的数据库本质上就是一个MongoDB数据库。MongoDB数据库是介于关系数据库和非关系数据库之间的产品,是非关系数据库中功能最丰富、最像关系数据库的。

数据库:默认情况下,云开发的函数可以使用当前环境对应的数据库。可以根据需要使用不同的数据库。对应MySQL中的数据库。

集合:数据库中多个记录的集合。对应MySQL中的表。

文档:数据库中的一条记录。对应MySQL中的行。

字段:数据库中特定记录的值。对应MySQL中的列。

3.1 云数据库上手

1. 创建第一个集合

打开云开发控制台,选择"数据库"标签页,通过单击集合名称右侧的"+"按钮创建一个集合。假设要创建一个设备查询类微信小程序,集合名称填写为device。创建成功后,可以看到device集合管理界面,在界面中可以添加记录、查找记录、管理索引和管理权限,如图3-1所示。

2. 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,选中集合device,单击"添加记录"按钮添加第一条设备数据,如图3-2所示。

图 3-1　创建集合

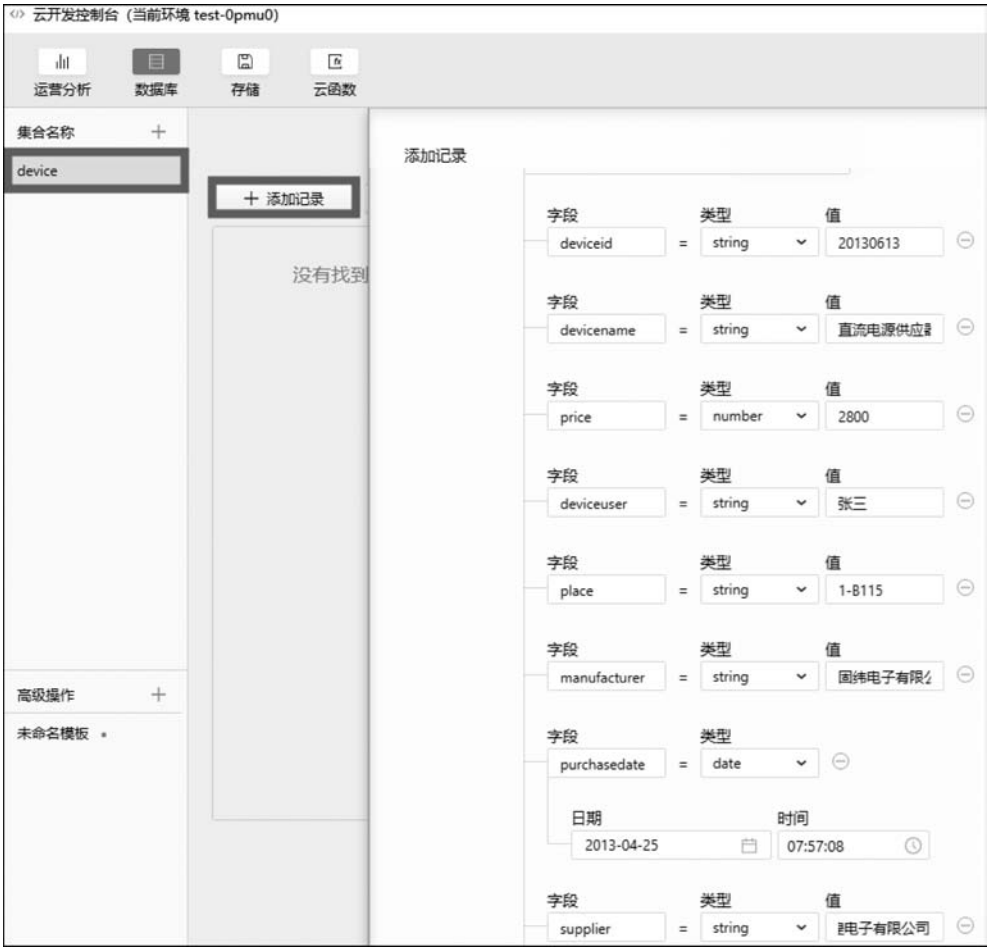

图 3-2　手动添加数据库记录

添加完成后可在控制台中查看到刚添加的数据，如图 3-3 所示。

图 3-3　数据库添加记录结果

3. 数据库高级操作

目前云开发控制台新增了数据库高级操作，如图 3-4 所示。

图 3-4　数据库高级操作

3.2　数据迁移

云开发支持从文件导入已有的数据。目前仅支持导入 CSV、JSON 格式的文件数据。有云开发控制台和 HTTP API 两种导入方式。

使用云开发控制台导入数据：要把文件导入云数据库，需打开云开发控制台，切换到“数据库”标签页，并选择要导入数据的集合，单击“导入”按钮，在“导入数据库”对话框中就可以将数据导入云数据库了，如图 3-5 所示。

选择要导入的 CSV 或者 JSON 文件，以及冲突处理模式，单击“确定”按钮即可开始导入。目前云数据库仅支持导入 CSV、JSON 格式的文件数据，有时数据通常以 Excel 的形式

图 3-5　将数据导入云数据库

出现，这里简单介绍 Excel 文件如何转换为 CSV 和 JSON 文件。

Excel 文件转换为 CSV 文件：打开 Excel 文件，然后单击“另存为”，保存类型选择“CSV(逗号分隔)(*.csv)”，因为数据库只支持 UTF-8 格式(默认为 ANSI 编码，导入后中文会显示成乱码)，然后用记事本再次打开 CSV 文件，另存为时选择编码为 UTF-8，如图 3-6 所示，替换原来的 CSV 文件即可。

图 3-6　用 CSV 格式保存为 UTF-8 编码

Excel 文件转换为 JSON 文件：开发者可以专门下载 Excel 转换为 JSON 工具，目前网站也提供了很多在线转换工具，如 http://www.bejson.com/json/col2json/，在线 Excel 文件转换为 JSON 文件的过程如图 3-7 所示。

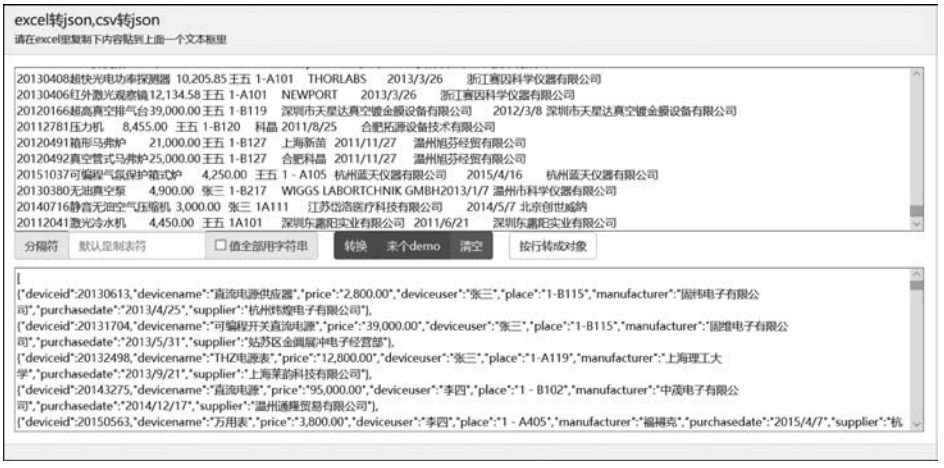

图 3-7　在线 Excel 文件转换为 JSON 文件的过程

然后把在线转换的结果复制到新建的 JSON 文件中，需要注意导入模板的 JSON 格式每条数据记录最后都没有逗号分隔，需要开发人员自行用文本替换把逗号去掉。数据导入结果如图 3-8 所示。

图 3-8　数据导入结果

需要注意以下几点：

（1）JSON 数据不是数组，而是类似 JSON Lines，即各个记录对象之间使用\n 分隔，而非逗号；

（2）JSON 数据每个键值对的键名首尾不能是“.”，例如“.a”“abc.”，且不能包含多个连续的“.”，例如“a..b”；

（3）键名不能重复，且不能有歧义，例如{"a":1, "a":2}或{"a": {"b":1},"a.b":2}；

（4）时间格式须为 ISODate 格式，例如"date":{"$date": "2018-08-31T17:30:00.882Z"}；

（5）当使用 Insert 冲突处理模式时，同一文件不能存在重复的_id 字段，或与数据库已有记录相同的_id 字段；

（6）CSV 格式的数据默认以第一行作为导入后的所有键名，余下的每一行则是与首行键名一一对应的键值记录。

目前提供了 Insert、Upsert 两种冲突处理模式。Insert 模式会在导入时总是插入新记录，Upsert 则会判断有无该条记录，如果有则更新记录，否则就插入一条新记录。

导入完成后，可以在提示信息中看到本次导入记录的情况。

如果开发人员不想通过在线转换工具（如 http://www.bejson.com/json/col2json/）进行转换，那么如何通过代码将 Excel 文件转换为 JSON 文件？本节通过引入依赖模块 xls-to-json 来实现将 Excel 文件转换为 JSON 文件，具体代码如下：

```
var node_xj = require("xls-to-json")
var fs = require('fs')
node_xj({
  input: "设备.xlsx",              // input xls
  output: "output.json",           // output json
  //sheet: "sheet1",               // specific sheetname
}, function (err, result) {
  if (err) {
    console.error(err)
  } else {
    console.log(result)
    console.log("JSON 文件已经被保存为 output.json.")
    fs.readFile('output.json', 'utf8', function (err, data) {
      if (err) {
        return console.log(err)
      }
      var result = data.replace(/},/g, '}\n').replace(/\[{/, '{').replace(/\}]/, '}')
      fs.writeFile('data.json', result, 'utf8', function (err) {
        if (err) return console.log(err)
      })
    })
  }
});
```

这里使用 VS Code 将本地的设备 Excel 文件转换成 JSON 文件，这里用到 xls-to-json 依赖库，读者使用之前需要在终端使用 npm install xls-to-json 安装此依赖库，代码第 3～7 行将 Excel 文件转换为 JSON 文件，输出文件名为 output.json，但是输出的 JSON 文件数据格式和图 3-7 一致，因此需要进一步修改格式。代码第 13～16 行读取 output.json 文件。代码第 17 行把最前面和最后面的[]去掉，并去掉每条记录后的逗号，最终生成云数据库可直接导入的 JSON 文件为 data.json；读者可以直接把 data.json 文件导入云数据库。

有时记录可能比较复杂，如图 3-9 中的 Excel 数据，其中的选项字段中需要用一个数组来存储，这些数据总不能让用户一条一条地输入数据库，那么这时就需要写代码将 Excel 数据记录转换为 JSON 文件。

针对图 3-9 中的 Excel 数据，Node.js 的代码如下：

```
var xlsx = require('node-xlsx')
var fs = require('fs')
var sheets = xlsx.parse('charpter1.xlsx')
var sheet = sheets[0].data
var fs = require('fs')
var json_data = []
for (var rowId = 1; rowId < sheet.length; rowId++) {
    let data = {}
    var row = sheet[rowId]
    data.type = row[0]
    data.title = row[1]
    data.option = row[2].split("\r\n");
```

题型	题干	选项	答案	解析	难易度
单选题	计算机所处理的数据一般具有某种内在联系，这是指（　　）。	A. 数据和数据之间存在某种关系 B. 数据元素和数据元素之间存在某种关系 C. 数据元素内部具有某种结构 D. 数据项和数据项之间存在某种关系	A		易
单选题	数据元素是数据的最小单位。这个断言是（　　）。	A. 正确的 B. 错误的	A		易
单选题	在数据结构中，与所使用的计算机无关的是数据的（　　）。	A. 逻辑结构 B. 存储结构 C. 逻辑结构和存储结构 D. 物理结构	A		易
单选题	数据的逻辑结构说明数据元素之间的次序关系，它依赖于数据的存储结构。这个断言是（　　）。	A. 正确的 B. 错误的	A		易
单选题	数据的存储结构是指数据在计算机内的实际存储形式。这个断言是（　　）。	A. 正确的 B. 错误的	A		易
单选题	顺序存储方式只能用于线性结构，不能用于非线性结构。这个断言是（　　）。	A. 正确的 B. 错误的	A		易
单选题	线性结构只能用顺序结构来存放，非线性结构只能用非顺序结构来存放。这个断言是（　　）。	A. 正确的 B. 错误的	A		易
单选题	计算机算法指的是（　　）。	A. 计算方法 B. 排序方法 C. 解决问题的有限运算序列 D. 调度方法	A		易
单选题	计算机算法必须具备输入、输出和（　　）等五个特性。	A. 可行性、可移植性和可扩充性 B. 可行性、确定性和有穷性 C. 确定性、有穷性和稳定性 D. 易读性、稳定性和安全性	A		易
单选题	下面关于算法的说法，正确的是（　　）。	A. 算法最终必须由计算机程序实现 B. 为解决某问题的算法同为该问题编写的程序含义是相同的 C. 算法的可行性是指指令不能有二义性 D. 以上几个都是错误的	A		易

图 3-9　Excel 数据

```
13      data.answer = row[3]
14      data.explain = row[4]
15      data.level = row[5]
16      json_data.push(data)
17  }
18  var json_str = JSON.stringify(json_data)
19  var result = json_str.replace(/},/g, '}\n').replace(/\[{/, '{').replace(/\}]/, '}')
20  console.log(result)
21  fs.writeFile('result.json', result, 'utf8', function (err) {
22      if (err) return console.log(err);
23  });
```

其中，代码第 3 行采用 node-xlsx 解析 charpter1.xlsx 文件，读者需要安装 node-xlsx 依赖库，在终端中输入 npm install node-xlsx；代码第 7～17 行读取每条记录，将每条记录转行成 JSON 对象，因为第一行对应表头，因此 rowId 从 1 开始，代码第 12 行通过 split("\r\n")方式，把字符串转换为数组；代码第 18 行将 JSON 对象转换为字符串；代码第 19 行把最前面和最后面的[]去掉，并去掉每条记录后的逗号；代码第 21～23 行把 JSON 字符串写入 result.json 文件。然后通过云开发控制台将 result.json 文件导入数据库，如图 3-10 所示。

图 3-10　JSON 文件导入数据库

3.3　基础概念

1. 数据类型

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。
- date：时间。
- geo：多种地理位置类型。
- null。

date 类型用于表示时间，精确到毫秒，在微信小程序端可用 JavaScript 内置 Date 对象创建。需要特别注意的是，在微信小程序端创建的时间是客户端时间，不是服务端时间，这意味着在微信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，该字段会被转换为服务端当前的时间，更棒的是，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，这样可以达到类似如下效果：指定一个字段为服务端时间往后一个小时。

如果需要使用客户端时间，存放 Date 对象和存放毫秒数效果是否一样呢？不一样，因为数据库有针对日期类型的优化，建议使用时都用 Date 或 serverDate 构造时间对象。

null 相当于一个占位符，表示一个字段存在但值为空。

2. 索引管理

建立索引是保证数据库性能、保证微信小程序体验的重要手段。开发人员应为所有需要成为查询条件的字段都建立索引。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。创建索引时可以指定增加唯一性限制，具有唯一性限制的索引会要求被索引集合不能存在被索引字段值都相同的两个记录。需特别注意的是，假如记录中不存在某个字段，则对索引字段来说其值默认为 null，如果索引有唯一性限制，则不允许存在两个或以上的该字段为空/不存在该字段的记录。在创建索引的时候索引属性选择“唯一”即可添加唯一性限制。

3. 权限控制

数据库的权限分为管理端和微信小程序端。管理端包括云函数端和云控制台。微信小程序端运行在微信小程序中，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。微信小程序端操作数据库应有严格的安全规则限制。

初期对操作数据库开放以下 4 种权限配置：仅创建者可写，所有人可读；仅创建者可读写；仅管理者可写，所有人可读；仅管理者可读写。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微信小程序深度整合，在微信小程序中创建的每个数据库记录都会带有该记录创建者（即微信小程序用户）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。

权限级别按照从宽到紧排列如下：

- 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，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，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，微信小程序端始终不能写他人创建的数据。微信小程序端的记录的读写权限其实分为：

（1）所有人可读，只有创建者可写；

（2）仅创建者可读写；

（3）所有人可读，仅管理端可写；

（4）所有人不可读，仅管理端可读写。

云数据库权限设置如表 3-1 所示。

表 3-1　云数据库权限设置

模　　式	微信小程序端读自己创建的数据	微信小程序端写自己创建的数据	微信小程序端读他人创建的数据	微信小程序端写他人创建的数据	管理端读写任意数据
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	×	×	×	×	√

集合权限设置如图 3-11 所示。在设置集合权限时应谨慎设置，防止出现越权操作。

图 3-11　集合权限设置

值得注意的是，新建集合的默认权限是：仅创建者可读写，如果是非创建者在微信小程序端是无法读取集合数据的。有些读者的数据是通过云开发控制台导入的，没有对集合的权限进行修改（默认权限是仅创建者可读写），就会出现在微信小程序读取不到集合数据的情况。

3.4　云数据库 API 列表

微信小程序云开发提供了丰富的数据库操作 API，数据库 API 都是“懒”执行的，这意味着只有真实需要网络请求的 API 调用才会发起网络请求，其余如获取数据库、集合、记录的引用、在集合上构造查询条件等都不会触发网络请求。

1. 请求

请求表示链式调用终结。触发网络请求的 API 有 get、add、update、set、remove 和 count，如表 3-2 所示。

表 3-2 触发网络请求的 API

API	说 明
get	获取集合/记录数据
add	在集合上新增记录
update	更新集合/记录数据
set	替换更新一个记录
remove	删除记录
count	统计查询语句对应的记录条数

2. 引用

获取引用的 API 有 database、collection 和 doc，如表 3-3 所示。

表 3-3 获取引用的 API

API	说 明
database	获取数据库引用，返回 Database 对象
collection	获取集合引用，返回 Collection 对象
doc	获取对一个记录的引用，返回 Document 对象

3. 数据库

数据库对象的字段有 command、serverDate 和 Geo，如表 3-4 所示。

表 3-4 数据库对象的字段

字 段	说 明
command	获取数据库查询及更新指令，返回 Command
serverDate	构造服务端时间
Geo	获取地理位置操作对象，返回 Geo 对象

4. 集合

集合对象 API 有 doc、add、where、orderBy、limit、skip 和 field，如表 3-5 所示。

表 3-5 集合对象 API

API	说 明
doc	获取对一个记录的引用，返回 Document 对象
add	在集合上新增记录
where	构建一个在当前集合上的查询条件，返回 Query，查询条件中可使用查询指令
orderBy	指定查询数据的排序方式
limit	指定返回数据的数量上限
skip	指定查询时从选中的记录列表中的第几项之后开始返回
field	指定返回结果中每条记录应包含的字段

5. 记录/文档

记录/文档对象 API 有 get、update、set、remove 和 field，如表 3-6 所示。

表 3-6　记录/文档对象 API

API	说　明
get	获取记录数据
update	局部更新数据
set	替换更新记录
remove	删除记录
field	指定返回结果中记录应包含的字段

6. 查询指令

Command 对象查询指令如表 3-7 所示。

表 3-7　Command 对象查询指令

类　别	指　令	说　明
比较	eq	字段是否等于指定值
	neq	字段是否不等于指定值
	lt	字段是否小于指定值
	lte	字段是否小于或等于指定值
	gt	字段是否大于指定值
	gte	字段是否大于或等于指定值
	in	字段值是否在指定数组中
	nin	字段值是否不在指定数组中
逻辑	and	条件与,表示需同时满足多个查询筛选条件
	or	条件或,表示只需满足其中一个条件即可
	nor	表示需所有条件都不满足
	not	条件非,表示对给定条件取反
字段	exists	字段存在
	mod	字段值是否符合给定取模运算
数组	all	数组所有元素是否满足给定条件
	elemMatch	数组是否有一个元素满足所有给定条件
	size	数组长度是否等于给定值
地理位置	geoNear	找出字段值在给定点的附近的记录
	geoWithin	找出字段值在指定区域内的记录
	geoIntersects	找出与给定的地理位置图形相交的记录

7. 更新指令

Command 对象更新指令如表 3-8 所示。

表 3-8　Command 对象更新指令

类　别	指　令	说　明
字段	set	设置字段为指定值
	remove	删除字段
	inc	原子操作,自增字段值
	mul	原子操作,自乘字段值

续表

类　别	指　令	说　明
	min	如果字段值小于给定值，则设为给定值
	max	如果字段值大于给定值，则设为给定值
	rename	字段重命名
数组	push	往数组尾部增加指定值
	pop	从数组尾部删除一个元素
	shift	从数组头部删除一个元素
	unshift	往数组头部增加指定值
	addToSet	原子操作，如果不存在给定元素则添加元素
	pull	剔除数组中所有满足给定条件的元素
	pullAll	剔除数组中所有等于给定值的元素

3.5　云数据库操作

从开发者工具 1.02.1906202 开始，在云控制台数据库管理页中可以编写和执行数据库脚本，脚本可对数据库进行 CRUD 操作，语法同 SDK 数据库语法，如图 3-12 所示。

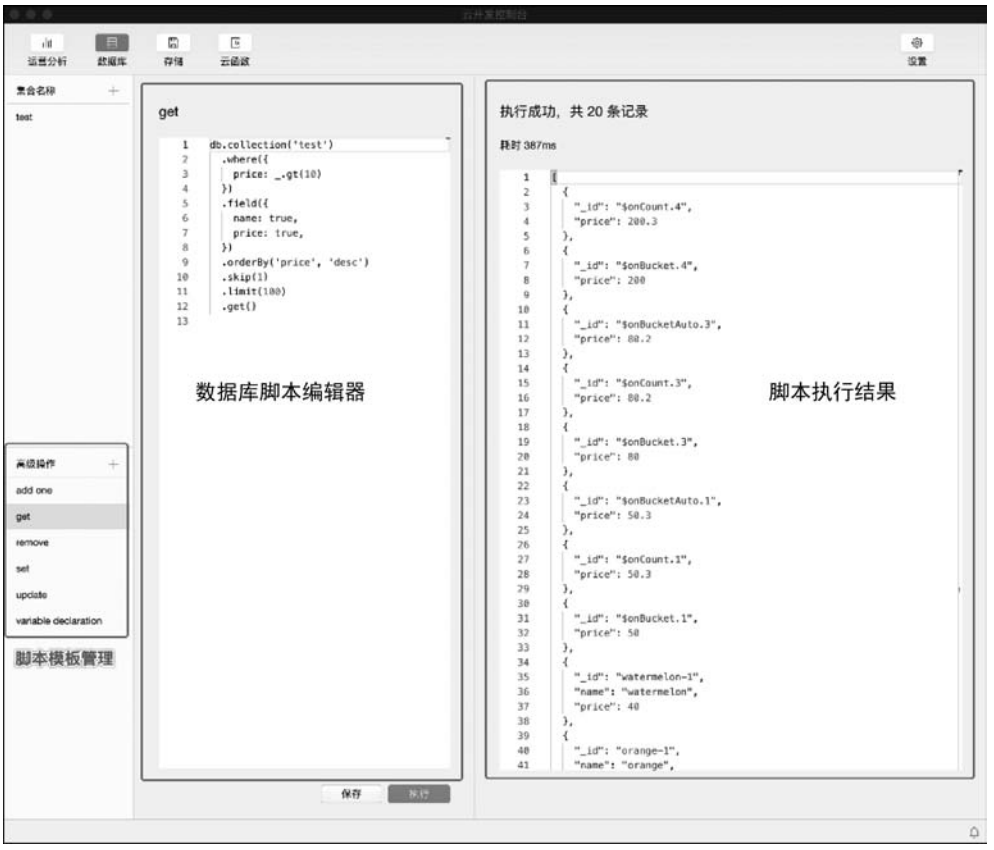

图 3-12　控制台编写和执行数据库脚本

3.5.1　增加记录

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，目前 add()方法一次只能插入一条记录。添加一条记录的执行语句为：

```
db.collection('device').add({
  data:{
    "deviceid":"20130410",
    "devicename":"红外狭缝扫描光束分析仪",
    "price":42042.29,
    "deviceuser":"王五",
    "place":"1-A101",
    "manufacturer":"THORLABS",
    "purchasedate":"2013/3/26",
    "supplier":"浙江赛因科学仪器有限公司"}
})
```

在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行数据库插入操作，结果如图 3-13 所示。

图 3-13　数据库插入操作

3.5.2　查询记录

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

1. 获取一个记录的数据

获取一个记录的数据执行语句为：

```
db.collection('device')
.doc('muRMVtf5R8RSu74VNS3juEdD7w3IRr2LJWXv6tQU4rZAXYrM')
.get()
```

在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行获取一个记录的数据，结果如图 3-14 所示。

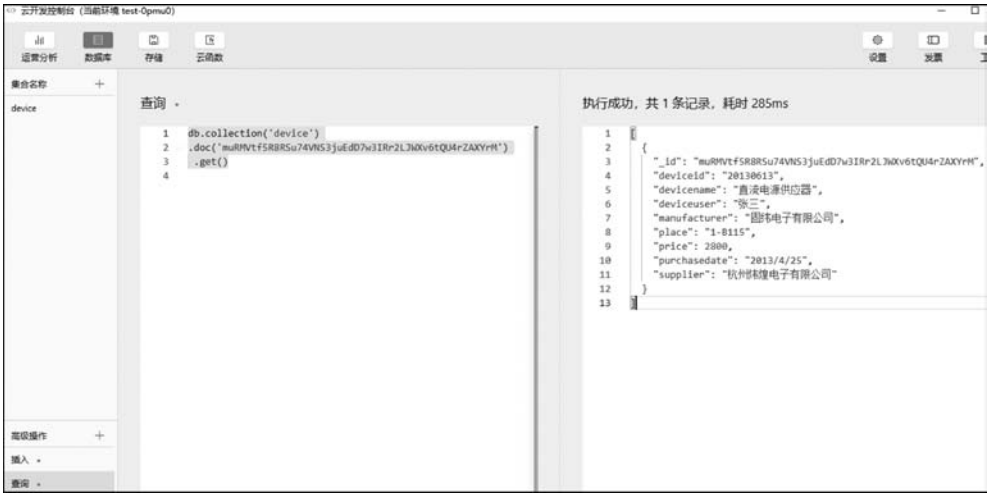

图 3-14　获取一个记录的数据

2. 获取多个记录的数据

当然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，比如获取用户张三名下所有的设备信息，获取多个记录的数据执行语句为：

```
db.collection('device').where({
  deviceuser:"张三"
})
.get()
```

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。在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行获取多个记录的数据，结果如图 3-15 所示。

3. 数据库查询指令

使用数据库 API 提供的 where()方法可以构造复杂的查询条件完成复杂的查询任务。API 提供的查询指令，如表 3-9 所示。

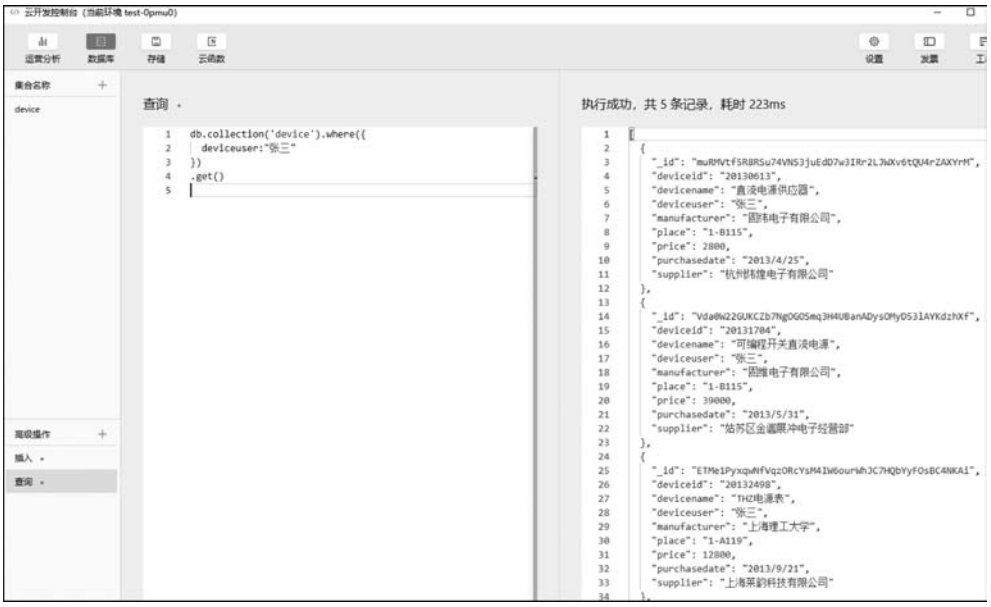

图 3-15　获取多个记录的数据

表 3-9　API 提供的查询指令

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

除了指定一个字段满足一个条件之外，还可以通过指定一个字段需同时满足多个条件，比如用 and 逻辑指令查询价格为 3500～10000 元的设备，执行代码为：

```
db.collection('device').where({
  price: _.gt(3500).and(_.lt(10000))
}).get()
```

在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行满足多个条件的查询，结果如图 3-16 所示。

既然有 and，当然也有 or 了，Command.or 用于表示逻辑“或”的关系，表示任意满足一个查询筛选条件。“或”指令有两种用法：一是可以进行字段值的“或”操作；二是可以进行跨字段的“或”操作。比如进行字段值的“或”操作，以查询设备为例，查询设备名称为“万用表”或“直流电源”的设备信息，执行代码为：

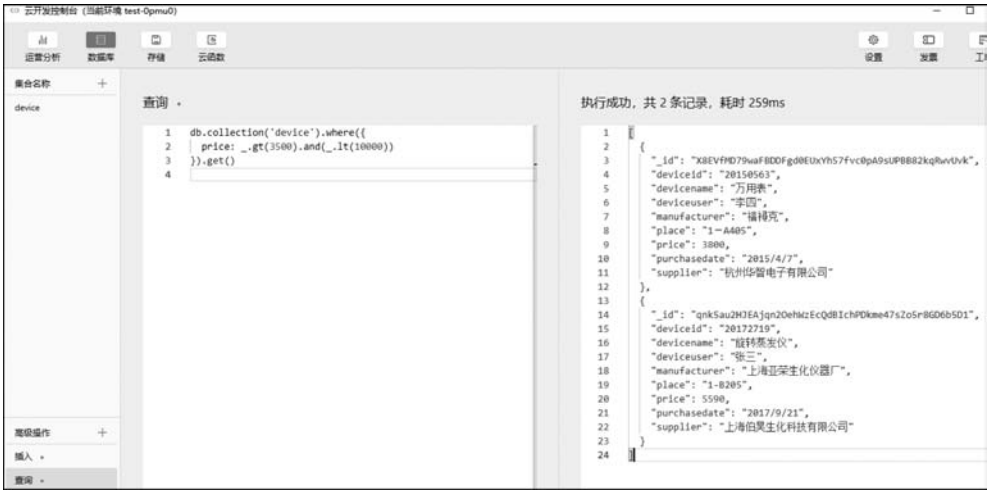

图 3-16　满足多个条件的查询

```
db.collection('device').where({
  devicename: _.eq('万用表').or(_.eq('直流电源'))
}).get()
```

在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行字段值“或”操作，结果如图 3-17 所示。

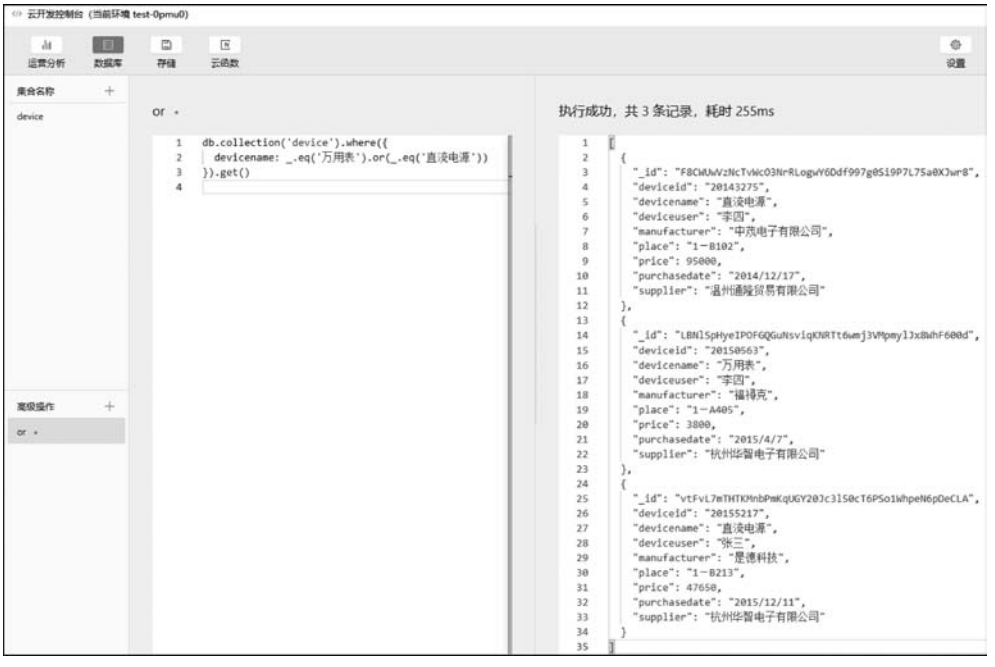

图 3-17　执行字段值“或”操作

还可以进行跨字段的“或”操作，以查询设备为例，查询设备名称为“万用表”或价格为 69300 元的设备信息，执行代码为：

```
db.collection('device').where(
  _.or([{
    devicename: _.eq('万用表')
  },
  {
    price: 69300
  }
  ])
)
  .get()
```

在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行跨字段“或”操作，结果如图 3-18 所示。

图 3-18　执行跨字段“或”操作

在一个实际的项目中，通常对数据库的操作需要用到集合上多个 API，接下来演示集合上多个 API 如何同时使用。首先用户在云数据库集合 device 中导入 124 条数据记录，用户在云开发控制台高级操作中执行命令：

```
db.collection('device')
  .where({
    price: _.gt(1000)
  })
  .field({
    deviceid: true,
    price: true,
  })
  .orderBy('price', 'desc')
  .skip(1)
  .limit(120)
  .get()
```

代码第 2～4 行中加入了查询的限制条件；代码第 5～8 行指定返回结果中每条记录应包含的字段；代码第 9 行指定查询数据的排序方式，其中 desc 表示降序，asc 表示升序；代码第 10 行指定查询时从选中的记录列表中的第几项之后开始返回；代码第 11 行指定返回数据的数量上限。在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行集合对象多个 API 查询，结果如图 3-19 所示。

图 3-19　集合对象多个 API 查询

这里有两点需要说明：

(1) API(add、where、orderBy、limit、skip 和 field 等)在调用时没有先后顺序；

(2) 在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行查询的返回记录没有数量限制，而在实际开发中微信小程序端在获取集合数据时服务器一次默认并且最多返回 20 条记录，云函数端这个数字则是 100。

4. 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 device 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，在微信小程序中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。开发者可以通过 limit()方法指定需要获取的记录数量，但微信小程序端不能超过 20 条，云函数端不能超过 100 条。如果要获取集合中所有记录的数据，很可能一个请求无法取出所有数据，需要分批次取。获取一个集合数据的执行代码为：

```
1  const cloud = require('wx-server-sdk')
2  cloud.init()
3  const db = cloud.database()
4  const MAX_LIMIT = 100
5  exports.main = async (event, context) =>{
6    let databasename = event.databasename
7    //先取出集合记录总数
8    const countResult = await db.collection(databasename).count()
9    const total = countResult.total
10   //计算需分几次取
```

```
11    const batchTimes = Math.ceil(total / 100)
12    //承载所有读操作的 promise 的数组
13    const tasks = []
14    for (let i = 0; i < batchTimes; i++) {
15      const promise = db.collection(databasename).skip(i * MAX_LIMIT).limit(MAX_LIMIT).get()
16      tasks.push(promise)
17    }
18    //等待所有
19    return (await Promise.all(tasks)).reduce((acc, cur) => {
20      return {
21        data: acc.data.concat(cur.data),
22        errMsg: acc.errMsg,
23      }
24    })
25  }
```

async 顾名思义是“异步”的意思，async 用于声明一个函数是异步的。而 await 从字面意思上是“等待”的意思，就是用于等待异步完成。也就是平常所说的异步等待。不过需注意 await 只能在 async 函数中使用。因此在云函数中使用 async-await()方法，可以把异步请求变为同步请求。代码第 6 行读取的从微信小程序调用云函数传递的数据库集合名称；第 8、9 行读取集合中记录总数；因为云函数读取数据库集合每次最多只能读取 100 条记录，因此代码第 11 行计算需要分几次读取集合中的数据；代码第 13～17 行分批读取集合中的数据。代码第 19～24 行把所有分批读取的数据进行汇总。

云函数的具体使用见第 5 章云函数的介绍，在微信小程序中调用云函数 database 代码如下：

```
1  wx.cloud.callFunction({
2    name: 'database',
3    data: {
4      databasename: "device"
5    }
6  }).then(res => {
7    console.log(res.result)
8  })
```

获取集合 device 的所有数据，结果如图 3-20 所示，通过云函数 database 把集合 device 中所有的 124 条数据都取出来了。

3.5.3　更新数据

更新数据主要有两个方法，如表 3-10 所示。

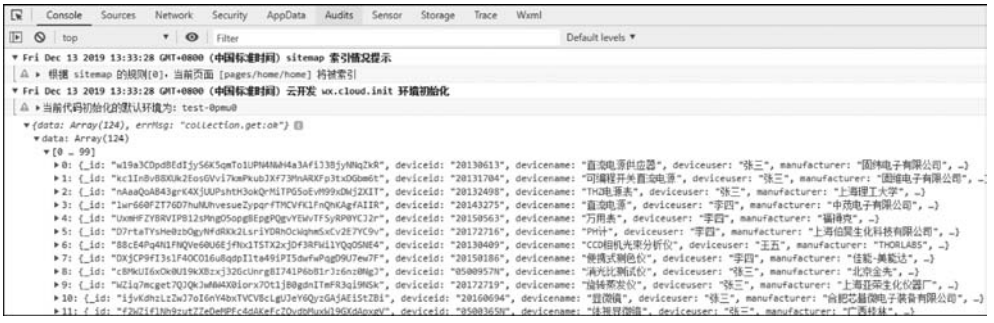

图 3-20　获取集合 device 的所有数据

表 3-10　更新数据的方法

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

1. 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。

局部更新执行代码为：

```
db.collection('device').doc('muRMVtf5R8RSu74VNS3juEdD7w3IRr2LJWXv6tQU4rZAXYrM')
  .update({
    data: {
      price: _.inc(100)
    }
  })
```

上面的示例代码演示了对设备某一条数据执行价格增加 100 的原子操作，用 inc 指令而不是取出值、加 100 再写进去的，其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 100 和 200 再写进数据库，那么这个字段最终结果会是加了 200 而不是 300。如果使用 inc 指令则不会有这个问题。在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行局部更新数据，结果如图 3-21 所示。

2. 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，执行代码为：

```
db.collection('device').doc('muRMVtf5R8RSu74VNS3juEdD7w3IRr2LJWXv6tQU4rZAXYrM')
  .set({
    data: {
      "deviceid":"20130613",
      "devicename":"直流电源供应器",
```

图 3-21　局部更新数据

```
        "price":2800.00,
        "deviceuser":"李四",
        "place":"1-B105",
        "manufacturer":"固纬电子有限公司",
        "purchasedate":"2013/5/25",
        "supplier":"杭州炜煌电子有限公司"
    }
})
```

使用上面的代码就会把指定的设备信息进行整个替换。

3.5.4　删除数据

对记录使用 remove()方法可以删除该条记录，比如：

```
db.collection('device').doc('83de3a0e-bcd0-483d-a197-402a5e1e80b5')
  .remove()
```

在云开发控制台中选择“数据库”标签页，在左侧树形目录中选择“高级操作”，执行完上面的数据库删除操作后，会把_id 为'83de3a0e-bcd0-483d-a197-402a5e1e80b5'这条记录进行删除操作。如果需要删除多条记录，则可在 Server 端进行操作(云函数)。

前面介绍了集合中数据的增、删、改、查操作，这些操作是通过在云开发控制台中进行演示，而微信小程序端，云函数端和控制台对数据库操作的权限是不同的，而且能够调用的 API 也是不同的，这里列举微信小程序端和云函数端对数据库调用 API 接口的不同之处，如表 3-11 所示。

表 3-11 微信小程序端和云函数端对数据库调用 API 接口的不同

类 别	Collection				Document			
	get	add	update	remove	get	add	update	remove
微信小程序端	√	√	×	×	√	——	√	√
云函数	√	√	√	√	√	——	√	√

从表中可以看出，针对 Document（单条记录操作），微信小程序端和云函数都支持查询、更新和删除操作，针对 Collection（批量记录操作），云函数支持批量查询、单条记录增加（目前一次只能添加一条记录）、批量更新和批量删除操作，但是微信小程序端只支持批量查询和单条记录增加操作，并不支持批量更新和批量删除操作。

3.5.5 正则表达式查询

数据库支持正则表达式查询，开发者可以在查询语句中使用 JavaScript 原生正则对象或使用 db.RegExp()方法来构造正则对象然后进行字符串匹配。在查询条件中对一个字段进行正则匹配即要求该字段的值可以被给定的正则表达式匹配。注意，正则表达式不可用于 db.command 内（如 db.command.in）。

db.RegExp()方法定义如下：

```
function RegExp(initOptions: IInitOptions): DBRegExp
interface IInitOptions {
  regexp: string                //正则表达式，字符串形式
  options: string               //flags，包括 i, m, s，但前端不做强限制
}
```

Options 支持 i、m 和 s 这 3 个 flag，注意 JavaScript 原生正则对象构造时仅支持其中的 i 和 m 两个 flag，因此需要使用 s 这个 flag 时必须使用 db.RegExp()构造正则对象。flag 的含义如表 3-12 所示。

表 3-12 flag 的含义

flag	说 明
i	大小写不敏感
m	跨行匹配：让开始匹配符^或结束匹配符 $ 除了匹配字符串的开头和结尾外，还匹配行的开头和结尾
s	让.可以匹配包括换行符在内的所有字符

首先用户在云数据库集合 device 中导入 124 条数据记录，用户可以在 regexp 参数中输入正则表达式（见图 3-22），查询 contact 集合中 mobile 字段以 138 开头、后面 8 个数字结尾的手机号记录，其中^表示匹配输入字行首，\d 表示匹配数字，$ 表示匹配输入行尾。

图 3-23 演示了查询 contact 集合中 name 字段是中文 4 个字的记录，其中^表示匹配输入字行首，$ 表示匹配输入行尾，“\u4e00”和“\u9fa5”是 Unicode 编码，并且正好是中文编码的开始和结束的两个值，所以这个正则表达式可以用来判断 name 包含 4 个中文的记录。

图 3-22　db. RegExp 正则匹配案例 1

图 3-23　db. RegExp 正则匹配案例 2

图 3-24 演示了查询 contact 集合中 name 字段以白字为结尾的记录，其中 $ 表示匹配输入行尾。

图 3-24　db. RegExp 正则匹配案例 3

上面都是在云开发控制台高级操作中进行演示，接下来以一个实际的通讯录查询案例演示使用正则表达式实现模糊查询。

ColorUI 给出了和手机通讯录类似的页面样式，用微信开发者工具打开 ColorUI 中的 demo 文件，在 tabBar 中选择“扩展”选项，然后单击“索引列表”图标，进入“索引”页面（代码见 pages/plugin/indexes），如图 3-25 所示。

为了便于看到通讯录的效果，本项目提前准备了 JSON 格式的数据（如何从 Excel 文件转换为 JSON 文件，见 3. 2 节），导入数据库 contact 集合，导入后的结果如图 3-26 所示，其中 searchfield 字段是手工加进去的，是为了便于对人员信息进行模糊搜索，实现用户在输入框中输入姓名的中文、拼音或者拼音首字母便可以搜索人员。

图 3-25　ColorUI 通讯录样式

图 3-26　通讯录数据集合

本案例的页面样式 home.wxml 代码如下：

```
<cu-custom bgColor="bg-gradual-pink" isBack="{{false}}">
  <view slot="backText">返回</view>
  <view slot="content">通讯录</view>
</cu-custom>

<view class="cu-bar bg-white search fixed" style="top:{{CustomBar}}px;">
  <view class="search-form round">
    <text class="cuIcon-search"></text>
    <input type="text" placeholder="输入名字" confirm-type="search"
bindinput="bindKeyInput"></input>
  </view>
  <view class="action">
    <button class="cu-btn bg-gradual-green shadow-blur round" bindtap='search'>搜索</button>
  </view>
</view>
<view class="cu-list menu-avatar no-padding">
  <block wx:for="{{contactCur}}" wx:key wx:for-index="sub">
    <view class="cu-item" bindtap="showModal" data-id="{{sub}}" data-target="bottomModal">
      <view class="cu-avatar round lg">{{item.sex}}</view>
      <view class="content">
        <view class="text-grey">{{item.name}}
        </view>
        <view class="text-gray text-sm">
          <text class="text-abc">{{item.mobile}}</text>
        </view>
      </view>
    </view>
  </block>
</view>
```

代码第 6～15 行对应搜索输入框；代码第 16～29 行对应搜索结果显示的样式，样式设计和 ColorUI 基本一致，包括用户性别、用户姓名、用户手机号。

为了显示样式，需要在 home.wxss 中加入如下代码：

```
page {
  padding-top: 100rpx;
}
```

相应的 home.js 代码实现如下：

```
const app = getApp()
const db = wx.cloud.database()
Page({
  data: {
    StatusBar: app.globalData.StatusBar,
    CustomBar: app.globalData.CustomBar,
    contactlist:[]
```

```
  },
  bindKeyInput: function (event) {
    const name = event.detail.value
    db.collection('contact').where({
      searchfield: db.RegExp({
        regexp: name,
        options: 'i',
      })
    }).get().then(res =>{
      this.setData({
        contactCur: res.data
      })
    })
  }
})
```

其中，bindKeyInput 对应输入框输入事件，代码第 11～16 行对应对 searchfield 字段进行模糊查询。对字段进行模糊查询的结果如图 3-27 所示，可以通过姓名的中文、拼音或者拼音首字母进行查询。

图 3-27　进行模糊查询的结果

3.5.6　查询和更新数组元素和嵌套对象

云数据库允许对对象、对象中的元素、数组、数组中的元素进行匹配查询，甚至还可以对数组和对象相互嵌套的字段进行匹配查询/更新，下面从普通匹配开始讲述如何进行匹配查

询/更新。

1. 普通匹配

传入的对象的每个< key, value >构成一个筛选条件,有多个< key, value >则表示需同时满足这些条件,是“与”的关系,如果需要“或”关系,可使用[command.or]。

比如找出未完成的进度 50 的待办事项:

```
db.collection('todos').where({
  done: false,
  progress: 50
}).get()
```

2. 匹配记录中的嵌套字段

假设在集合中有如下记录:

```
{
  "style": {
    "color": "red"
  }
}
```

如果想要找出集合中 style.color 为 red 的记录,那么可以传入相同结构的对象做查询条件或使用“点表示法”查询:

```
//方式一
db.collection('todos').where({
  style: {
    color: 'red'
  }
}).get()
//方式二
db.collection('todos').where({
  'style.color': 'red'
}).get()
```

3. 匹配数组

假设在集合中有如下记录:

```
{
  "numbers": [10, 20, 30]
}
```

可以传入一个完全相同的数组来筛选出这条记录:

```
db.collection('todos').where({
  numbers: [10, 20, 30]
}).get()
```

4. 匹配数组中的元素

如果想找出数组字段中数组值包含某个值的记录，可以在匹配数组字段时传入想要匹配的值。如对上面的例子，可传入一个数组中存在的元素来筛选出所有 numbers 字段的值包含 20 的记录：

```
db.collection('todos').where({
  numbers: 20
}).get()
```

5. 匹配数组第 n 项元素

如果想找出数组字段中数组的第 n 个元素等于某个值的记录，那么在< key, value >匹配中可以以“字段.下标”为 key，目标值为 value 做匹配。如对上面的例子，如果想找出 number 字段第二项的值为 20 的记录，查询如下(注意，数组下标从 0 开始)：

```
db.collection('todos').where({
  'numbers.1': 20
}).get()
```

更新也是类似，比如要更新_id 为 test 的记录的 numbers 字段的第二项元素至 30：

```
db.collection('todos').doc('test').update({
  data: {
    'numbers.1': 30
  },
})
```

6. 结合查询指令进行匹配

在对数组字段进行匹配时，也可以使用如 lt、gt 等指令来筛选出字段数组中存在满足给定比较条件的记录。如对上面的例子，可查找出所有 numbers 字段的数组值中存在包含大于 25 的值的记录：

```
const _ = db.command
db.collection('todos').where({
  numbers: _.gt(25)
}).get()
```

查询指令也可以通过逻辑指令组合条件，比如找出所有 numbers 数组中存在包含大于 25 的值同时也存在小于 15 的值的记录：

```
const _ = db.command
db.collection('todos').where({
  numbers: _.gt(25).and(_.lt(15))
}).get()
```

7. 匹配并更新数组中的元素

如果想要匹配并更新数组中的元素，而不是替换整个数组，除了指定数组下标外，还可以：

(1) 更新数组中第一个匹配到的元素。

更新数组字段时可以用字段路径.$ 的表示法来更新数组字段的第一个满足查询匹配条件的元素。注意，使用这种更新时，查询条件必须包含该数组字段。

假如有如下记录：

```
{
  "_id": "doc1",
  "scores": [10, 20, 30]
}
{
  "_id": "doc2",
  "scores": [20, 20, 40]
}
```

让所有 scores 中的第一个 20 的元素更新为 25：

```
//注意:批量更新需在云函数中进行
const _ = db.command
db.collection('todos').where({
  scores: 20
}).update({
  data: {
    'scores.$': 25
  }
})
```

如果记录是对象数组也可以做到，路径如字段路径.$.字段路径。

注意事项：

- 不支持用在数组嵌套数组；
- 如果用 unset 更新操作符，不会从数组中去除该元素，而是置为 null；
- 如果数组元素不是对象，且查询条件用了 neq、not 或 nin，则不能使用.$。

(2) 更新数组中所有匹配的元素。

更新数组字段时可以用字段路径.$[]的表示法来更新数组字段的所有元素。

假如有如下记录：

```
{
  "_id": "doc1",
  "scores": {
    "math": [10, 20, 30]
  }
}
```

比如让 scores.math 字段中所有数字加 10：

```
const _ = db.command
db.collection('todos').doc('doc1').update({
  data: {
    'scores.math.$[]': _.inc(10)
  }
})
```

更新后 scores.math 数组从[10，20，30]变为[20，30，40]。

如果数组是对象数组也是可以的。假如有如下记录：

```
{
  "_id": "doc1",
  "scores": {
    "math": [
      { "examId": 1, "score": 10 },
      { "examId": 2, "score": 20 },
      { "examId": 3, "score": 30 }
    ]
  }
}
```

可以更新 scores.math 下各个元素的 score 原子自增 10：

```
const _ = db.command
db.collection('todos').doc('doc1').update({
  data: {
    'scores.math.$[].score': _.inc(10)
  }
})
```

8. 匹配多重嵌套的数组和对象

上面所讲述的所有规则都是可以嵌套使用的。假设在集合中有如下记录：

```
{
  "root": {
    "objects": [
      {
        "numbers": [10, 20, 30]
      },
      {
        "numbers": [50, 60, 70]
      }
    ]
  }
}
```

下面的查询语句找出集合中所有满足 root.objects 字段数组的第二项的 numbers 字段的第三项等于 70 的记录：

```
db.collection('todos').where({
  'root.objects.1.numbers.2': 70
}).get()
```

注意，指定下标不是必需的，例如可以找出集合中所有满足 root.objects 字段数组中任意一项的 numbers 字段包含 30 的记录：

```
db.collection('todos').where({
  'root.objects.numbers': 30
}).get()
```

更新操作也是类似，例如要更新_id 为 test 的 root.objects 字段数组的第二项的 numbers 字段的第三项为 80：

```
db.collection('todos').doc('test').update({
  data: {
    'root.objects.1.numbers.2': 80
  },
})
```

3.5.7　数据库操作 data 赋值

数据库操作中通常需要设置 data 的值，data 的值本质上是 JSON 格式的对象，JOSN 格式的对象在 2.1.3 节中进行了介绍，JSON 格式的对象必须遵守如下写法：对象被花括号{}包围；对象以键值对书写；键必须是字符串，值必须是有效的 JSON 数据类型；键和值由冒号分隔；每个键值对由逗号分隔。

例如，常见的添加记录的方式为(需要在云数据库添加 device 集合)：

```
addData: function(e) {
  const db = wx.cloud.database()
  db.collection('device').add({
    data: {
      "deviceid": "20130613",
      "devicename": "直流电源供应器",
      "price": 2800.00,
      "deviceuser": "张三",
      "place": "1-B115",
      "manufacturer": "固纬电子有限公司",
      "purchasedate": new Date('2013-4-25'),
      "supplier": "杭州炜煌电子有限公司",
      "submitdata":db.serverDate()
    }
    })
```

```
    .then(res =>{
      console.log(res)
    })
}
```

代码第 7 行 price 字段的类型为数字，第 11 行和第 13 行字段类型为 date 类型，第 13 行获取服务器时间。调用 addData 事件，在数据库中 device 集合中插入一条设备记录，如图 3-28 所示。

图 3-28　微信小程序端插入数据库记录

有时在插入数据库之前已经有 JSON 对象了，那么是不是也一定需要用键值对的方式写 data 呢？既然 data 的值本质上是 JSON 格式的对象，就可以直接把 JSON 格式的对象赋值给 data，例如：

```
addData: function(e) {
  var device1 = {
    "deviceid": "20130613",
    "devicename": "直流电源供应器",
    "price": 2800.00,
    "deviceuser": "张三",
    "place": "1-B115",
    "manufacturer": "固纬电子有限公司",
    "purchasedate": new Date('2013-4-25'),
    "supplier": "杭州炜煌电子有限公司"
  }
  const db = wx.cloud.database()
  device1.submitdata = db.serverDate()
  db.collection('device').add({
    data: device1
    })
```

```
    .then(res =>{
      console.log(res)
    })
}
```

上面例子实现的结果和图 3-28 一致。

此外，data 数据中的字段还支持对象和数组类型。接下来以数组为例演示如何在 vote 集合中插入一条投票记录(需要在云数据库添加 vote 集合)：

```
addData: function (e) {
    var options = []
    options[0] = {
      option: 'A: 800 元以下',
      num: 0
    }
    options[1] = {
      option: 'B: 800 元至 1000 元',
      num: 0
    }
    options[2] = {
      option: 'C: 1000 元至 2000 元',
      num: 0
    }
    options[3] = {
      option: 'D: 2000 元以上',
      num: 0
    }
    const db = wx.cloud.database()
    db.collection('vote').add({
      data: {
        "title": "投票标题：你平均一月用去多少生活费",
        "type": "单选题",
        options
      }
    })
      .then(res =>{
        console.log(res)
      })
  }
```

代码第 21～25 行的 data 数据中，插入了 options 数组，插入数据库后该字段的字段名为数组的变量名，插入云数据库后的结果如图 3-29 所示。

前面演示了设置对象字段的值，那么如果需要访问的字段名是一个变量，在 data 中如何进行访问呢？接下来以上面插入的投票记录为例，演示如何更新其中一个字段的值，例如：

```
addData: function (e) {
    var opt = '1'
    var options = 'options.' + opt + '.num'
    const db = wx.cloud.database()
    const _ = db.command
    console.log(options)
    db.collection('vote').doc('72527ac65dfd84f1056f295e5b6df49e').update({
      data: {
        [options]: _.inc(1)
      }
    })
      .then(console.log)
      .catch(console.error)
  }
```

图 3-29 data 字段值为数组

为了演示，代码第 7 行记录的_id 值来源于图 3-29 中的记录，这里假设用户在实际投票页面选择了 B 选项，相应地需要把投票记录中 options[1].num 的值自增 1，在这里相应票数增加采用原子操作 db.command.inc，难点在于票数都在 options 字段中，而 options 本身是个数组，对于更新数组元素的操作读者可以参考 3.5.6 节，这里使用“点表示法”，例如 options.1.num 表示 options 数组中第一个元素中的 num 的值。在这里读者需要注意，data 中的字段名是变量，需要把变量放入[]中，调用 addData 事件以后 vote 集合中相应的选项 num 会增加 1。操作后数据库中的结果如图 3-30 所示。

3.5.8 增、删、改、查案例

接下来以设备数据为例，演示云数据库增、删、改、查的方法。

选择样式：用微信开发者工具打开 ColorUI 中的 demo 文件，在 tabBar 中选择“扩展”选项，然后单击“垂直导航”图标，进入“Tab 索引”页面，ColorUI 中设备查询条目样式如图 3-31 所示。

图 3-30　data 字段名是变量

图 3-31　ColorUI 中设备查询条目样式

在 ColorUI 中打开 plugin/verticalnav/verticalnav. wxml 页面找到对应代码，选取如下：

```
1  <view class = "cu - list menu - avatar">
2    <view class = "cu - item">
3      <view class = "cu - avatar round lg"
4  style = " background - image: url(https://ossweb - img. qq. com/images/lol/web201310/skin/
5  big10001. jpg);"></view>
```

```
    <view class="content">
      <view class="text-grey">凯尔</view>
      <view class="text-gray text-sm flex">
        <text class="text-cut">
        <text class="cuIcon-infofill text-red
margin-right-xs"></text>我以天理为凭,踏入这片荒芜,不再受凡人的枷锁遏制。我以天理为
凭,踏入这片荒芜,不再受凡人的枷锁遏制。
        </text>
      </view>
    </view>
    <view class="action">
      <view class="text-grey text-xs">22:20</view>
      <view class="cu-tag round bg-grey sm">5</view>
    </view>
  </view>
</view>
```

开发者可以直接使用上面的样式,也可以对上面的样式进行适当的修改,修改后的样式如下(background-image 的图片从 https://www.iconfont.cn 中搜索设备图标下载后,上传至云存储):

```
<view class="cu-list menu sm-border margin-top">
  <block wx:for="{{deviceslist}}" wx:key="index">
    <view class="cu-item">
      <view class="cu-avatar round lg"
style="background-image:url(https://7465-test-0pmu0-1300559272.tcb.qcloud.la/device/
shebei.png?sign=000998c6a7a5ad83e6a677b71d7bf10a&t=1573090290);"></view>
      <view class="content">
        <view class="text-grey">
          <text class="text-lg text-grey">设备名称:</text>
          <text class="text-lg text-mauve">{{item.devicename}}</text>
        </view>
        <view>
          <text class="text-lg text-grey">设备编号:</text>
          <text class="text-lg text-mauve">{{item.deviceid}}</text>
          <text class="text-lg text-grey">领用人:</text>
          <text class="text-lg text-mauve">{{item.deviceuser}}</text>
        </view>
      </view>
      <view class="action">
        <button class="cu-btn bg-green margin-tb-sm ">编辑</button>
      </view>
    </view>
  </block>
</view>
```

在微信小程序页面加载时进行设备查询,进入微信小程序微信官方文档(https://developers.weixin.qq.com/miniprogram/dev/framework/),选择“云开发”子页面,在左侧

树形目录中选择“SDK 文档”→“数据库”→Collection→get 选项，选择数据库集合查询语句，如图 3-32 所示。

图 3-32　数据库集合查询语句

复制图 3-32 中的数据库集合查询语句到 home.js 文件中的 onLoad 事件中，并在 data 中加入 deviceslist 对象用于在 home.wxml 中显示，具体如下：

```
1  data: {
2      deviceslist: {},
3    },
4  onLoad: function(options) {
5    const db = wx.cloud.database()
6    db.collection('device').get().then(res =>{
7      //console.log(res.data)
8      this.setData({ deviceslist: res.data })
9    })
10 }
```

采用 Collection.get 获取了多条数据库记录，如图 3-33 所示。

图 3-33　数据库多条数据集合查询结果

接下来演示如何根据条件进行数据查询。首先选择搜索框的样式：用微信开发者工具打开 ColorUI 中的 demo 文件，在 tabBar 中选择“扩展”选项，然后单击“索引列表”图标，进入“索引”页面，选择的搜索框样式如图 3-34 所示。

图 3-34　选择的搜索框样式

相应的代码目录为 plugin/indexes/indexes.wxml，选取代码加入 home.wxml 页面，代码如下：

```
<cu-custom bgColor="bg-gradual-pink" isBack="{{false}}">
  <view slot="backText">返回</view>
  <view slot="content">设备清单</view>
</cu-custom>
<view class="cu-bar bg-white search fixed" style="top:{{CustomBar}}px;">
  <view class="search-form round">
    <text class="cuIcon-search"></text>
    <input type="text" placeholder="输入设备编号" bindinput="bindKeyInput"></input>
  </view>
  <view class="action">
    <button class="cu-btn bg-gradual-green shadow-blur round" bindtap='deviceSearch'>搜索</button>
  </view>
</view>
<view class="cu-list menu sm-border margin-top" style="padding-top: 100rpx;">
  <block wx:for="{{deviceslist}}" wx:key="index">
    <view class="cu-item">
      <view class="cu-avatar round lg"
style="background-image:url(https://7465-test-0pmu0-1300559272.tcb.qcloud.la/device/
shebei.png?sign=000998c6a7a5ad83e6a677b71d7bf10a&t=1573090290);"></view>
      <view class="content">
        <view class="text-grey">
          <text class="text-lg text-grey">设备名称:</text>
          <text class="text-lg text-mauve">{{item.devicename}}</text>
        </view>
        <view>
          <text class="text-lg text-grey">设备编号:</text>
          <text class="text-lg text-mauve">{{item.deviceid}}</text>
          <text class="text-lg text-grey">领用人:</text>
          <text class="text-lg text-mauve">{{item.deviceuser}}</text>
        </view>
      </view>
      <view class="action">
        <button class=" cu-btn bg-green margin-tb-sm" bindtap="recordEdit" data-
id="{{item._id}}">编辑</button>
      </view>
```

```
    </view>
  </block>
</view>
```

相应地，本案例增加了输入框的输入事件，以及“搜索”按钮的搜索事件，在该事件中，根据用户在输入框输入的设备 ID(deviceid)，对云数据库进行数据查询，相应的代码如下：

```
const app = getApp();
const db = wx.cloud.database()
Page({
  data: {
    StatusBar: app.globalData.StatusBar,
    CustomBar: app.globalData.CustomBar,
    deviceslist: {},
    deviceid: '',
  },
  onLoad: function(options) {
    db.collection('device').get().then(res =>{
      this.setData({ deviceslist: res.data })
    })
  },
  bindKeyInput: function (event) {
    this.setData({
      deviceid: event.detail.value
    })
  },
  deviceSearch:function(event){
    var deviceid = this.data.deviceid
    if (deviceid != '') {
      db.collection('device').where({
        deviceid: deviceid
      }).get().then(res =>{
        this.setData({ deviceslist: res.data })
      })
    }
    else
    {
      db.collection('device').get().then(res =>{
        this.setData({ deviceslist: res.data })
      })
    }
  },
  recordEdit:function(event)
  {
    wx.navigateTo({
      url: '../edit/edit? _id=' + event.currentTarget.dataset.id
    })
  },
})
```

根据条件进行数据查询，页面效果如图 3-55 所示。代码第 20～35 行实现了设备记录的查询功能，当输入框中有设备编号时(输入框不为空)，对该设备编号进行查询；如果输入框为空，则查找所有设备记录。在这里为了便于演示数据库的操作，查找所有设备记录时，实际上微信小程序只能查找 20 条数据，读者需要在 onReachBottom 事件(触底刷新事件)中，使用数据库 Collection.skip 加载后续的设备记录，该功能在后续章节案例中进行讲解。

接下来演示对单条数据进行更新和删除操作。添加页面 pages/edit/edit，选择表单样式。用微信开发者工具打开 ColorUI 中的 demo 文件，在 tabBar 中选择“组件”选项，然后单击“表单”，进入“表单”页面，单条设备数据编辑样式选择如图 3-36 所示。

图 3-35 根据条件进行数据查询

图 3-36 单条设备数据编辑样式选择

相应的代码目录为 componet/form/form.wxml，选取代码加入 edit.wxml 页面，代码如下：

```
1  <cu-custom bgColor="bg-gradual-pink" isBack="{{true}}">
2    <view slot="backText">返回</view>
3    <view slot="content">设备编辑</view>
4  </cu-custom>
5
6  <form>
7    <view class="cu-form-group margin-top">
8      <view class="title">
9        <text style="color:red">*</text>设备编号:</view>
10     <input placeholder="请输入设备编号" value='{{deviceinfo.deviceid}}' data-name='deviceid'
```

```
bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>设备名称:</view>
    <input placeholder = "请输入设备名称" value = '{{deviceinfo.devicename}}' data - name =
'devicename' bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>设备领用人:</view>
    <input placeholder = "请输入设备领用人" value = '{{deviceinfo.deviceuser}}'
data - name = 'deviceuser' bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>设备产商:</view>
    <input placeholder = "请输入设备产商" value = '{{deviceinfo.manufacturer}}'
data - name = 'manufacturer' bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>设备存放地:</view>
    <input placeholder = "请输入设备存放地" value = '{{deviceinfo.place}}' data - name = 'place'
bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>设备价格:</view>
    <input placeholder = "请输入设备价格" value = '{{deviceinfo.price}}' data - name = 'price'
bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>购买日期:</view>
    <input placeholder = "请输入购买日期" value = '{{deviceinfo.purchasedate}}'
data - name = 'purchasedate' bindinput = "updateValue"></input>
  </view>
  <view class = "cu - form - group">
    <view class = "title">
      <text style = "color:red">*</text>供应商:</view>
    <input placeholder = "请输入供应商" value = '{{deviceinfo.supplier}}' data - name = 'supplier'
bindinput = "updateValue"></input>
  </view>
</form>

<view class = "flex padding justify - center">
  <button class = "cu - btn lg bg - pink" bindtap = "deviceupdate">修改设备</button>
  <button class = "cu - btn lg bg - pink" bindtap = "deviceDelete">删除设备</button>
</view>
```

相应的 edit.js 的完整代码如下：

```
const db = wx.cloud.database()
Page({
  data: {
    deviceinfo: {}
  },
  onLoad: function(options) {
    const _id = options._id
    db.collection('device').doc(_id).get().then(res =>{
      this.setData({
        deviceinfo: res.data
      })
    })
  },

  updateValue: function(event) {
    let name = event.currentTarget.dataset.name;
    let deviceinfo = this.data.deviceinfo
    deviceinfo[name] = event.detail.value
    this.setData({
      deviceinfo: deviceinfo
    })
  },

  deviceupdate: function(event) {
    let deviceinfo = this.data.deviceinfo
    db.collection('device').doc(deviceinfo._id).set({
      data: {
        deviceid: deviceinfo.deviceid,
        devicename: deviceinfo.devicename,
        deviceuser: deviceinfo.deviceuser,
        manufacturer: deviceinfo.manufacturer,
        place: deviceinfo.place,
        price: parseInt(deviceinfo.price),
        purchasedate: deviceinfo.purchasedate,
        supplier: deviceinfo.supplier
      },
      success: function(res) {
        if (res.stats.updated) {
          console.log("更新成功...")
        }
      }
    })
  },

  deviceDelete: function(event) {
    let deviceinfo = this.data.deviceinfo
    db.collection('device').doc(deviceinfo._id).remove()
      .then(res =>{
```

```
49          console.log("删除成功...")
50          wx.navigateTo({
51            url: '../home/home'
52          })
53        })
54    },
55  })
```

代码第6～13行实现页面加载(onLoad事件)时,根据上一个页面传过来的参数_id,查询数据库中的记录;代码第15～22行实现页面中输入框输入事件的监听;代码第24～43行实现数据库记录的更新;代码第45～55行实现数据库记录的删除操作。

在home页面单击设备记录后面的"编辑"按钮,就会进入edit页面,edit页面样式如图3-37所示。

图3-37　edit页面样式

需要说明的是,本案例在云开发控制台中设置集合device的权限设置为"所有用户可读,仅创建者可读写"。由于数据记录是通过导入的方式添加进集合,因此每条数据都没有记录创建者。如果是通过微信小程序添加的记录,就会在每条记录中添加一个_openid字段,而通过数据导入的方式不会自动添加_openid字段,因此在edit页面中直接单击"修改设备"按钮,会提示类似的错误:

```
1  Error: errCode: -502001 database request fail | errMsg: [FailedOperation.set] multiple
2  write errors: [{write errors: [{E11000 duplicate key error collection: tnt-12p3936xo.x-
3  j-1 index: id dup key: { : "xjl" }}]}, {<nil>}]
```

造成上面这种错误的主要原因是这条数据不是用户自己创建的，因为集合 device 只支持创建者可读写。要解决这个问题，有两种方法：

（1）在每条记录中添加_openid 字段，值为开发者自己的 openid，但是该方法也仅支持创建者对该记录的写操作；

（2）通过云函数，所有用户都可对该记录进行读写。

在这里通过添加_openid 字段的方式，如图 3-38 所示。

图 3-38　数据添加_openid 字段

_openid 的值可以通过云函数获得，也可以进入云开发控制台，选择“运营分析”→“用户访问”选项，直接复制 openid。