第5章

云开发中小程序端数据库开发

本章先介绍数据类型、权限控制、初始化等概念，再介绍如何在小程序端向集合中插入数据、查询数据、使用查询指令、更新数据、使用更新指令、删除数据，接着介绍在小程序端对集合的其他操作方法、正则表达式的用法、如何处理地理信息 db.Geo、聚合的用法等内容。因此本章的 API 主要指小程序端 API。

5.1 基础概念

5.1.1 数据类型

云开发数据库提供 String(字符串)、Number(数字)、Object(对象)、Array(数组)、Boolean(布尔值)、Date(时间)、Geo(多种地理位置类型)、Null 等几种数据类型。Null 相当于一个占位符，表示一个字段存在但是值为空。在官方文档中，数据类型的首字母有时用小写字母(如 string)，有时用大写字母(如 String)，本书统一用大写字母，除非为了特别强调才会用到小写字母。

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，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，这样就可以指定一个字段为服务端时间往后 offset 毫秒。

当需要使用客户端时间时，存放 Date 对象和存放毫秒数的效果不是一样的，由于数据库有针对日期类型的优化，使用时用 Date 或 serverDate 构造时间对象效果更好。

要使用地理位置查询功能时，必须建立地理位置索引，建议用于存储地理位置数据的字段均建立地理位置索引。可以在云控制台建立地理位置索引。

云开发数据库提供了多种地理位置数据类型的增加、删除、查询、修改支持，支持的地理位置数据类型包括 Point(点)、LineString(线段)、Polygon(多边形)、MultiPoint(点集合)、MultiLineString(线段集合)和 MultiPolygon(多边形集合)。

5.1.2 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。

初期对操作数据库开放 4 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条记录上的。出于易用性和安全性的考虑，云开发为数据库做了小程序深度整合，在小程序中创建的数据库每条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。

按照对数据库的操作权限级别从宽到紧排列如下：

(1) 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，例如文章。

(2) 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，例如用户私密相册。

(3) 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，例如商品信息。

(4) 仅管理端可读写：该数据只有管理端可读写，例如后台用的不能暴露的数据。

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。

5.1.3 初始化

在开始使用数据库 API 进行数据的增加、删除、查询、修改操作之前，需要先获取数据库的引用。如果需要获取对其他环境中数据库的引用，则可以在调用 wx.cloud.database() 方法时传入一个对象参数，在其中通过 env 字段指定要使用的环境。

操作一个集合，也需要先获取它的引用。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。获取集合的引用并不会发起网络请求拉取它的数据，可以通过此引用在该集合上进行数据的增加、删除、查询、修改操作，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。

5.2　在小程序端向集合中插入数据

视频讲解

5.2.1　API 说明

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。add()方法的参数 options 是必填参数，对象 options 的字段信息如表 5-1 所示，如果传入参数包括 success、fail、complete 3 个字段中的任何一个，则表示使用回调风格，不会返回 Promise。

表 5-1　options 的字段信息

字　段　名	类　　型	必　填	说　　明
data	Object	是	新增记录的定义
success	Function	否	成功回调，回调传入的参数 Result 包含查询的结果
fail	Function	否	失败回调
complete	Function	否	调用结束的回调函数(调用成功、失败都会执行)

如传入的 options 参数没有 success、fail、complete 3 个字段中的任何一个，则返回一个 Promise，否则不返回任何值。Promise 的结果包括 resolve(即新增记录的结果 Result)和 reject(即失败原因)。options 参数 success 回调的结果 Result 和 Promise 中 resolve 的结果 Result 都是一个新增的记录的 ID，该 ID 可以是 String 或 Number 类型。

5.2.2　辅助工作

按照 2.4.2 节的方法，在目录 D:\wxxyx\secondcloud 中创建一个小程序云开发项目 secondcloud。

按照 3.2.1 节的方法，在环境 learnwxbookscode(环境 ID 为 learnwxbookscode-wsd001)中创建一个数据库集合 mpcloudbook。

5.2.3　修改文件 app.json

修改文件 app.json，文件 app.json 修改后的代码如例 5-1 所示。代码的修改方法是在语句“"pages/index/index",”之前增加语句“"pages/insertData/insertData",”。后面碰到同类型的修改，不再给出文件 app.json 的完整代码，只说明代码的修改方法。

【例 5-1】　文件 app.json 修改后的代码示例。

```
{
  "pages": [
    "pages/insertData/insertData",
    "pages/index/index",
    "pages/userConsole/userConsole",
    "pages/storageConsole/storageConsole",
    "pages/databaseGuide/databaseGuide",
```

```
      "pages/addFunction/addFunction",
      "pages/deployFunctions/deployFunctions",
      "pages/chooseLib/chooseLib",
      "pages/openapi/openapi",
      "pages/openapi/serverapi/serverapi",
      "pages/openapi/callback/callback",
      "pages/openapi/cloudid/cloudid",
      "pages/im/im",
      "pages/im/room/room"
    ],
    "window": {
      "backgroundColor": "#F6F6F6",
      "backgroundTextStyle": "light",
      "navigationBarBackgroundColor": "#F6F6F6",
      "navigationBarTitleText": "云开发 QuickStart",
      "navigationBarTextStyle": "black"
    },
    "sitemapLocation": "sitemap.json"
}
```

修改代码后编译程序，自动在目录 pages 下生成 insertData 子目录，且在 pages/insertData 目录下自动生成了 insertData 页面的 4 个文件(如 insertData.wxml 等)。

5.2.4 修改文件 insertData.wxml

修改文件 insertData.wxml，文件 insertData.wxml 修改后的代码如例 5-2 所示。

【例 5-2】 文件 insertData.wxml 修改后的代码示例。

```
<!-- pages/insertData/insertData.wxml -->
<text>pages/insertData/insertData.wxml</text>
<button type="primary" bindtap="insertOneRecord">插入一条记录</button>
<button type="primary" bindtap="insertRecordPromise">以 Promise 方式插入一条记录
</button>
```

5.2.5 修改文件 insertData.js

修改文件 insertData.js，文件 insertData.js 修改后的代码如例 5-3 所示。

【例 5-3】 文件 insertData.js 修改后的代码示例。

```
//pages/insertData/insertData.js
Page({
  insertOneRecord: function() {
    //获取默认环境中数据库的引用
    const db = wx.cloud.database()
    //获取集合的引用后增加数据
```

```
    db.collection('mpcloudbook').add({
      //data 字段表示需新增的 JSON 数据
      data: {
        //_id: 'todo-identifiant-aleatoire-3', //数据库自动分配也可自定义
        title: "微信小程序开发基础",
        description: "对微信小程序开发进行入门性、基础介绍.",
        author: "woodstone",
        publishDate: new Date("2018-09-01"),
        topics: [
          "mini program",
          "database",
          "spring boot"
        ],
        //徐州地理位置(117°E,34°N)
        location: new db.Geo.Point(117, 34),
        //是否已经出版
        published: true
      },
      success: function(res) {
        console.log("成功插入一条记录.")
      },
      fail: console.error
    })
  },
  insertRecordPromise: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').add({
        data: {
          title: "Spring Boot 开发实战",
          description: "learn Spring Boot",
          author: "woodstone",
          publishDate: new Date("2018-09-01"),
          topics: [
            "Spring Boot",
            "NoSQL",
            "Thymeleaf",
            "Restful"
          ],
          location: new db.Geo.Point(113, 23),
          published: true
        }
      })
      .then(res => {
        console.log("用 Promise 成功插入一条记录.")
      })
      .catch(console.error)
  }
})
```

5.2.6 运行程序

编译程序,模拟器中的输出结果如图 5-1 所示。依次单击图 5-1 中的“插入一条记录”按钮和“以 Promise 方式插入一条记录”按钮,控制台中的输出结果如图 5-2 所示。与此同时,云数据库集合 mpcloudbook 中增加两条记录,结果如图 5-3 所示。

图 5-1 编译程序后在模拟器中的输出结果

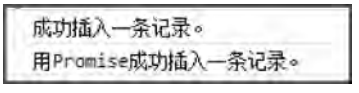

图 5-2 依次单击图 5-1 中两个按钮后控制台的输出结果

图 5-3 云开发控制台中数据库集合 mpcloudbook 中增加记录的结果

5.3 在小程序端查询数据

视频讲解

5.3.1 API 说明

在记录和集合上都提供了 get()方法用于获取单条记录或集合中多条记录的数据。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。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，各个字段间是逻辑“与”的关系，即需要同时满足这些匹配条件。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。

如果要获取一个集合的数据，例如获取集合上的所有记录，可以在集合上调用 get()方法获取，但通常不建议这么使用，在小程序中要尽量避免一次性获取过量的数据，应只获取必要的数据。为了防止误操作和保护小程序的使用体验，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，云函数端在获取集合数据时服务器一次默认并且最多返回 100 条记录。开发者可以通过 limit()方法指定需要获取的记录数量，但不能超过默认的上限。

5.3.2　辅助工作

在 5.2 节项目 secondcloud 和数据库集合 mpcloudbook 的基础上继续后续的开发。

往集合 mpcloudbook 中插入多条记录，以备查询。

修改文件 app.json，代码的修改方法是在语句“"pages/insertData/insertData",”之前增加语句“"pages/getData/getData",”。修改代码后编译程序，自动在目录 pages 下生成 getData 子目录，且在 pages/getData 目录下自动生成了 getData 页面的 4 个文件（如 getData.wxml 等）。

5.3.3　修改文件 getData.wxml

修改文件 getData.wxml，文件 getData.wxml 修改后的代码如例 5-4 所示。

【例 5-4】 文件 getData.wxml 修改后的代码示例。

```
<!-- pages/getData/getData.wxml -->
<text>pages/getData/getData.wxml</text>
<button type="primary" bindtap="getAllData">获得所有数据</button>
<button type="primary" bindtap="getAllDataPromise">以 Promise 方式获得所有数据</button>
<button type="primary" bindtap="getOneRecord">获取一条记录</button>
<button type="primary" bindtap="getRecordPromise">以 Promise 方式获取一条记录</button>
<button type="primary" bindtap="getRecords">获取多条记录</button>
<button type="primary" bindtap="getRecordsConditions">多个条件查询</button>
<button type="primary" bindtap="getDataDot">点表示法查询记录</button>
<button type="primary" bindtap="getDataPage">分页取数据</button>
```

5.3.4　修改文件 getData.js

修改文件 getData.js，文件 getData.js 修改后的代码如例 5-5 所示。

【例 5-5】 文件 getData.js 修改后的代码示例。

```
//pages/getData/getData.js
Page({
  getAllData: function() {
    const db = wx.cloud.database()
    //get 方法会触发网络请求,往数据库取数据
    db.collection('mpcloudbook').get({
      success(res) {
        console.log(res)
      }
    })
  },
  getAllDataPromise: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').get().then(res => {
      //res.data 是一个集合中有权限访问的所有记录的数据,小程序端不超过 20 条
      console.log(res.data)
    })
  },
  getOneRecord: function() {
    const db = wx.cloud.database()
    //获取记录的引用后查询数据
    db.collection('mpcloudbook').doc('Spring Boot').get({
      success: function(res) {
        console.log(res.data)
      }
    })
  },
  getRecordPromise: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').doc('Spring Boot').get().then(res => {
      console.log(res.data)
    })
  },
  //使用 where()方法传入一个对象
  getRecords: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
        _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI',     //改成开发者自己的_openid
        published: true
      })
      .get({
        success: function(res) {
          console.log(res.data)
        }
      })
  },
```

```
  getRecordsConditions: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
        _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI',
        type: {
          bookclassnum: 'tn929'
        }
      })
      .get({
        success: function(res) {
          console.log(res.data)
        }
      })
  },
  getDataDot: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
        _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI',
        'type.bookclassnum': 'tn929'
      })
      .get({
        success: function(res) {
          console.log(res.data)
        }
      })
  },
  getDataPage: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
        _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI', //填入当前用户 openid
      })
      .skip(5)       //跳过结果集中的第一页(前 5 条)
      .limit(5)      //从第 6 条开始返回,限制返回数量为 5 条(即第二页)
      .get()
      .then(res => {
        console.log(res.data)
      })
      .catch(err => {
        console.error(err)
      })
  }
})
```

5.3.5　运行程序

编译程序,模拟器中的输出结果如图 5-4 所示。从顶部向底部依次单击图 5-4 中的 8 个按钮,控制台中的输出结果如图 5-5 所示。

图 5-4　编译程序后模拟器中的输出结果

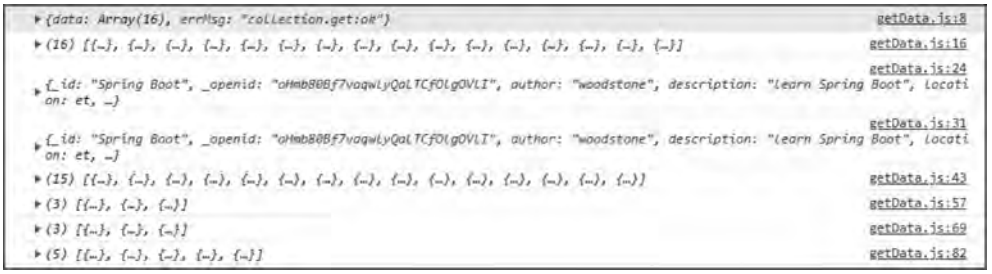

图 5-5　从顶部向底部依次单击图 5-4 中的 8 个按钮后控制台中的输出结果

5.3.6　运行程序后控制台中 JSON 结果数据的检验说明

由于本书中示例的数据内容较多、动态变化，而且一些数据内容（例如，数据库的_id、文件路径等）是自动生成的，再加上本书由于篇幅的原因描述的细节不是面面俱到（例如，除非有必要，否则书中只会说明增加一条记录而不会说明增加记录的具体取值），这样就会让读者在实践本书应用开发示例时输入、生成的数据和本书示例的数据库、存储内容数据不完全一致（数据的不一致不会影响到读者的学习）。于是读者在实践时获得的返回记录数量或返回数据细节（如数据库的_id）有较大的可能性和本书的结果不一致。另外，在控制台中展开返回的 JSON 数据细节会导致结果的截图内容太多且会有较多的重复，以图 5-5 的第 1、2 行为例，就各自成功获得了 16 条 JSON 数据（由于其他辅助信息的存在两者返回结果细节的行数总和超过 32 行，且两者的核心数据相同），这将会使得本书的篇幅大大增加。

为了节约篇幅，并考虑到读者的云开发中数据库、存储内容和书上的数据库、存储内容可能有差异，本书对控制台中 JSON 结果数据的截图中没有给出数据细节。程序成功运行后返回的 JSON 结果数据中包括返回 JSON 数据总数信息，例如图 5-5 中第 1 行结果中的

“Array(16)”和第 2 行结果中的“(16)”。有时由于没有符合条件要求的数据返回，返回 JSON 结果数据为空([])，这时返回结果中包括 ok 这样的返回信息(如图 5-5 所示的第 1 行结果)。请读者在实践书中示例时，注意用是否返回了 JSON 数据总数信息、是否存在 ok 这样的返回信息、是否存在在代码中设置的成功返回信息(三者只要有一个出现即可)来判断程序是否运行成功，而不要以返回的记录数量或数据细节来判断程序是否运行成功。例如，读者在实践本节例子时返回的 JSON 数据总数(如图 5-5 所示的第 1 行结果)为“Array(10)”也说明运行程序成功了，而不一定要求返回如图 5-5 中第 1 行结果中的“Array(16)”。另外，读者还可以结合视频来加深对示例程序运行成功与否的认识。

5.4　在小程序端使用查询指令

视频讲解

5.4.1　API 说明

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。API 提供的常用查询指令如表 5-2 所示。

表 5-2　API 提供的常用查询指令

查 询 指 令	说　　明
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
and	条件与，表示需同时满足另一个条件
or	条件或，表示如果满足另一个条件也匹配
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
exists	字段存在
mod	字段值是否符合给定取模运算
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

5.4.2　辅助工作

在 5.3 节项目 secondcloud 和数据库集合 mpcloudbook 的基础上继续后续的开发。

修改文件 app.json，代码的修改方法是在语句“"pages/getData/getData",”之前增加语句“"pages/dbcommandex/dbcommandex",”。修改代码后编译程序，自动在目录 pages 下

生成 dbcommandex 子目录，且在 pages/dbcommandex 目录下自动生成了 dbcommandex 页面的 4 个文件(如 dbcommandex. wxml 等)。

5.4.3 修改文件 dbcommandex. wxml

修改文件 dbcommandex. wxml，文件 dbcommandex. wxml 修改后的代码如例 5-6 所示。

【例 5-6】 文件 dbcommandex. wxml 修改后的代码示例。

```
<!-- pages/dbcommandex/dbcommandex.wxml -->
<text>pages/dbcommandex/dbcommandex.wxml</text>
<button type="primary" bindtap="dbOneComEx">一个条件查询指令</button>
<button type="primary" bindtap="dbAndComsEx">与逻辑查询指令</button>
<button type="primary" bindtap="dbOrComsEx">或逻辑查询指令</button>
<button type="primary" bindtap="dbComHaveEx">字段取值的条件指令</button>
<button type="primary" bindtap="dbComEqualEx">等值条件指令</button>
<button type="primary" bindtap="dbComExistsEx">exists 指令</button>
<button type="primary" bindtap="dbComAllEx">all 指令</button>
```

5.4.4 修改文件 dbcommandex. js

修改文件 dbcommandex. js，文件 dbcommandex. js 修改后的代码如例 5-7 所示。

【例 5-7】 文件 dbcommandex. js 修改后的代码示例。

```
//pages/dbcommandex/dbcommandex.js
Page({
  dbOneComEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').where({
        //gt()方法用于指定一个"大于"条件
        price: _.gt(30)     //查询价格高于 30 的书籍
      })
      .get({
        success: function(res) {
          console.log(res.data)
        }
      })
  },
  dbAndComsEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').where({
        //and()方法用于指定一个"与"条件，此处表示需同时满足两个条件
        price: _.gt(30).and(_.lt(50))
      })
```

```
      .get({
        success: function(res) {
          console.log(res.data)
        }
      })
  },
  dbOrComsEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    //or 表示逻辑或运算
    db.collection('mpcloudbook').where(_.or([{
          price: _.lte(49)
        },
        {
          type: {
            bookclassnum: _.in(['tn929', 'tp311'])
          }
        }
      ]))
      .get({
        success: function(res) {
          console.log(res.data)
        }
      })
  },
  dbComHaveEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').where({
        price: _.in([49, 49.8])
      })
      .get({
        success: function(res) {
          console.log(res.data)
        },
        fail: console.error
      })
  },
  dbComEqualEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').where({
      type: _.eq({
        bookclassnum: 'tn929'
      })
    }).get({
      success: function(res) {
        console.log(res.data)
      },
```

```
        fail: console.error
      })
    },
    dbComExistsEx: function() {
      const db = wx.cloud.database()
      const _ = db.command
      db.collection('mpcloudbook').where({
        published: _.exists(true)
      }).get({
        success: function(res) {
          console.log(res.data)
        },
        fail: console.error
      })
    },
    dbComAllEx: function() {
      const db = wx.cloud.database()
      const _ = db.command
      db.collection('mpcloudbook').where({
        topics: _.all(['NoSQL'])
      }).get({
        success: function(res) {
          console.log(res.data)
        },
        fail: console.error
      })
    }
  })
```

5.4.5 运行程序

编译程序,模拟器中的输出结果如图 5-6 所示。从顶部向底部依次单击图 5-6 中的 7 个按钮,控制台中的输出结果如图 5-7 所示。

图 5-6 编译程序后在模拟器中的输出结果

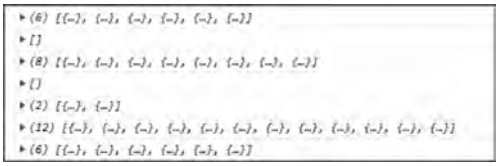

图 5-7　从顶部向底部依次单击图 5-6 中 7 个按钮后控制台中的输出结果

5.5　在小程序端更新数据和使用更新指令

视频讲解

5.5.1　API 说明

数据库 API 中更新数据有 update()、set()两种方法。使用 update()方法可以局部更新一条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。当需要替换更新一条记录时，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。

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(如 db.command.set 指令)。常用更新指令的信息如表 5-3 所示。

表 5-3　常用更新指令的信息

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
mul	原子自乘字段值
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
push	如果字段值为数组，则往数组尾部增加指定值
pop	如果字段值为数组，则从数组尾部删除一个元素
shift	如果字段值为数组，则从数组头部删除一个元素
unshift	如果字段值为数组，则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

5.5.2　辅助工作

在 5.4 节项目 secondcloud 和数据库集合 mpcloudbook 的基础上继续后续的开发。

修改文件 app.json，代码的修改方法是在语句“"pages/dbcommandex/dbcommandex",”之前增加语句“"pages/updatedata/updatedata",”。修改代码后编译程序，自动在目录 pages

中生成 updatedata 子目录，且在 pages/updatedata 目录中自动生成了 updatedata 页面的 4 个文件（如 updatedata.wxml 等）。

5.5.3 修改文件 updatedata.wxml

修改文件 updatedata.wxml，文件 updatedata.wxml 修改后的代码如例 5-8 所示。

【例 5-8】 文件 updatedata.wxml 修改后的代码示例。

```
//pages/updatedata/updatedata.js
Page({
  dbUpdateEx: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').doc('Spring Boot').update({
      //data 传入需要局部更新的数据
      data: {
        price: 95
      },
      success: function(res) {
        console.log('成功更新数据.')
      }
    })
  },
  dbUpdatePromise: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot1').update({
        data: {
          author: "zhangsanfeng"
        }
      })
      .then(console.log("Promise 方式成功更新数据."))
      .catch(console.error)
  },
  updateCommandExample: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        //表示将字段自增 10
        price: _.inc(10)
      },
      success: function(res) {
        console.log('成功用 update 指令更新数据.')
      },
      fail: console.error
    })
  },
  updateArrayEx: function() {
```

```
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        topics: _.push('MySQL')
      },
      success: function(res) {
        console.log("成功更新数组中数据.")
      }
    })
  },
  updateObjectEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        type: _.set({
          bookclassnum: 'tp313'
        })
      },
      success: function(res) {
        console.log("成功用 set 指令更新对象数据.")
      }
    })
  },
  updateExchangeEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring1').set({
      data: {
        description: "change to Spring Boot",
        publishDate: new Date("2018 - 09 - 01"),
        topics: [
          "cloud",
          "database"
        ]
      },
      success: function(res) {
        console.log("替换更新成功.")
      }
    })
  },
  updateMinEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        price: _.min(150)
      },
      success: function(res) {
```

```
          console.log("数据更新成功.")
        }
      })
    },
    updateAddToSetEx: function() {
      const db = wx.cloud.database()
      const _ = db.command
      db.collection('mpcloudbook').doc('Spring Boot').update({
        data: {
          topics: _.addToSet({
            each: ['database', 'cloud']
          })
        },
        success: function(res) {
          console.log("补充数据成功.")
        }
      })
    },
})
```

5.5.4 修改文件 updatedata.js

修改文件 updatedata.js，文件 updatedata.js 修改后的代码如例 5-9 所示。

【例 5-9】 文件 updatedata.js 修改后的代码示例。

```
//pages/updatedata/updatedata.js
Page({
  dbUpdateEx: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').doc('Spring Boot').update({
      //data 传入需要局部更新的数据
      data: {
        price: 95
      },
      success: function(res) {
        console.log('成功更新数据.')
      }
    })
  },
  dbUpdatePromise: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot1').update({
        data: {
          author: "zhangsanfeng"
        }
      })
```

```
      .then(console.log("Promise 方式成功更新数据."))
      .catch(console.error)
  },
  updateCommandExample: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        //表示将字段自增 10
        price: _.inc(10)
      },
      success: function(res) {
        console.log('成功用 update 指令更新数据.')
      },
      fail: console.error
    })
  },
  updateArrayEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        topics: _.push('MySQL')
      },
      success: function(res) {
        console.log("成功更新数组中数据.")
      }
    })
  },
  updateObjectEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        type: _.set({
          bookclassnum: 'tp313'
        })
      },
      success: function(res) {
        console.log("成功用 set 指令更新对象数据.")
      }
    })
  },
  updateExchangeEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
```

```
    db.collection('mpcloudbook').doc('Spring1').set({
      data: {
        description: "change to Spring Boot",
        publishDate: new Date("2018-09-01"),
        topics: [
          "cloud",
          "database"
        ]
      },
      success: function(res) {
        console.log("替换更新成功.")
      }
    })
  },
  updateMinEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        price: _.min(150)
      },
      success: function(res) {
        console.log("数据更新成功.")
      }
    })
  },
  updateAddToSetEx: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('Spring Boot').update({
      data: {
        topics: _.addToSet({
          each: ['database', 'cloud']
        })
      },
      success: function(res) {
        console.log("补充数据成功.")
      }
    })
  },
})
```

5.5.5 运行程序

编译程序,模拟器中的输出结果如图 5-8 所示。从顶部向底部依次单击图 5-8 中的 8 个按钮,控制台中的输出结果如图 5-9 所示。

图 5-8　编译程序后模拟器中的输出结果

图 5-9　从顶部向底部依次单击图 5-8 中 8 个按钮后控制台中的输出结果

5.6　在小程序端删除数据

视频讲解

5.6.1　API 说明

对一条记录使用 remove()方法可以删除该条记录，也可以用更新指令 db.command.remove 删除某个字段。remove()方法的参数 options 是必填参数，对象 options 信息如表 5-4 所示，如果传入了 success、fail、complete 3 个字段中的任意一个，则表示使用回调风格，不返回 Promise。

表 5-4　对象 options 信息

字　段　名	类　　型	必　填	说　　明
data	Object	是	新增记录的定义
success	Function	否	回调成功，回调传入的参数 Result 包含查询的结果
fail	Function	否	回调失败
complete	Function	否	调用结束的回调函数(调用成功、失败都会执行)

如果传入的 options 参数没有 success、fail、complete 3 个字段中的任意一个，则返回一个 Promise，否则不返回任何值。Promise 的结果包括 resolve(即新增记录的结果 Result)和 reject(即失败原因)。options 参数中 success 回调的结果 Result 及 Promise 中 resolve 的结果 Result 是一个更新结果的统计的 stats 对象(Object 类型)。stats 对象只有一个 Number 类型的 remove 字段，用来表示成功删除的记录数量，取值只能为 0 或者 1。

5.6.2　辅助工作

在 5.5 节项目 secondcloud 和数据库集合 mpcloudbook 的基础上继续后续的开发。

修改文件 app.json，代码的修改方法是在语句“"pages/dbcommandex/dbcommandex",”之

前增加语句“""pages/deleteData/deleteData",”。修改代码后编译程序，自动在目录 pages 下生成 deleteData 子目录，且在 pages/deleteData 目录下自动生成了 deleteData 页面的 4 个文件（如 deleteData.wxml 等）。

5.6.3 修改文件 deletedata.wxml

修改文件 deletedata.wxml，文件 deletedata.wxml 修改后的代码如例 5-10 所示。

【例 5-10】 文件 deletedata.wxml 修改后的代码示例。

```
<!-- pages/deleteData/deleteData.wxml -->
<text>pages/deleteData/deleteData.wxml</text>
<button type="primary" bindtap="deleteaField">删除一个字段</button>
<button type="primary" bindtap="deleteOneRecord">删除一条记录</button>
<button type="primary" bindtap="deleteRecordPromise">Promise 方式删除一条记录</button>
```

5.6.4 修改文件 deletedata.js

修改文件 deletedata.js，文件 deletedata.js 修改后的代码如例 5-11 所示。

【例 5-11】 文件 deletedata.js 修改后的代码示例。

```
//pages/deleteData/deleteData.js
Page({
  deleteaField: function() {
    const db = wx.cloud.database()
    const _ = db.command
    db.collection('mpcloudbook').doc('123434515d9461d7090f30527ad3534b').update({
      data: {
        pubilshed: _.remove()
      },
      success: function(res) {
        console.log("成功用 remove 指令删除一个字段.")
      }
    })
  },
  deleteOneRecord: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').doc('123434515d9461d7090f30527ad3534b').remove({
      success: function(res) {
        console.log("成功删除一条记录.")
      },
      fail: console.error
    })
  },
  deleteRecordPromise: function() {
    const db = wx.cloud.database()
```

```
      db.collection('mpcloudbook').doc('1af3506e5d94612a090ef6e01207c567').remove({
        success: function(res) {
          console.log("成功用 Promise 方式删除一条记录.")
        },
        fail: console.error
      })
    }
})
```

5.6.5　运行程序

编译程序，模拟器中的输出结果如图 5-10 所示。从顶部向底部依次单击图 5-10 中的 3 个按钮，控制台中的输出结果如图 5-11 所示。

图 5-10　编译程序后模拟器中的输出结果

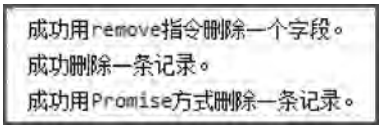

图 5-11　从顶部向底部依次单击图 5-10 中 3 个按钮后控制台的输出结果

5.7　在小程序端对集合的其他操作方法

视频讲解

5.7.1　API 说明

count()方法可以统计集合中记录数或统计查询语句对应的结果记录数。注意，此方法与集合的权限设置有关，一个用户仅能统计其有读权限的集合记录数。

orderBy()方法接收一个必填字符串参数 fieldName 用于定义需要排序的字段，一个字符串参数 order 定义排序顺序。order 只能取 asc 或 desc。如果需要对嵌套字段排序，则需要用“点表示法”连接嵌套字段，如 style.color。同时也支持按多个字段排序，多次调用 orderBy()即可，多字段排序时的顺序会按照 orderBy()调用顺序先后对多个字段排序。

field()方法接收一个必填对象用于指定要返回的字段，对象的各个 key 表示要返回或不要返回的字段，value 传入 true 或 false(或者 1 或－1)表示要返回还是不要返回。

watch()方法监听集合中符合查询条件的数据的更新事件。使用 watch()时，只有 where 语句会生效，orderBy、limit 等语句不会生效。其中，snapshot 信息如表 5-5 所示。ChangeEvent 信息如表 5-6 所示。QueueType 信息如表 5-7 所示。DataType 信息如表 5-8 所示。

表 5-5 snapshot 信息

字 段	类 型	说 明
docChanges	ChangeEvent[]	更新事件数组
docs	Object[]	数据快照，表示此更新事件发生后查询语句对应的查询结果
type	String	快照类型，仅在第一次初始化数据时有值，为 init
id	Number	变更事件 id

表 5-6 ChangeEvent 信息

字 段	类 型	说 明
id	Number	更新事件 id
queueType	String	列表更新类型，表示更新事件对监听列表的影响，枚举值，定义见 QueueType
dataType	String	数据更新类型，表示记录的具体更新类型，枚举值，定义见 DataType
docId	String	更新的记录 id
doc	Object	更新的完整记录
updatedFields	Object	所有更新的字段及字段更新后的值，key 为更新的字段路径，value 为字段更新后的值，仅在 update 操作时有此信息
removedFields	String[]	所有被删除的字段，仅在 update 操作时有此信息

表 5-7 QueueType 信息

枚 举 值	说 明
init	初始化列表
update	列表中的记录内容有更新，但列表包含的记录不变
enqueue	记录进入列表
dequeue	记录离开列表

表 5-8 DataType 信息

枚 举 值	说 明
init	初始化数据
update	更新记录，对应 update 操作
replace	替换记录，对应 set 操作
add	新增记录，对应 add 操作
remove	删除记录，对应 remove 操作

5.7.2　辅助工作

在5.6节项目secondcloud和数据库集合mpcloudbook的基础上继续后续的开发。

修改文件app.json，代码的修改方法是在语句“"pages/dbcommandex/dbcommandex",”之前增加语句“"pages/otherCollectionMethods/otherCollectionMethods",”。修改代码后编译程序，自动在目录pages下生成otherCollectionMethods子目录，且在pages/otherCollectionMethods目录下自动生成了otherCollectionMethods页面的4个文件（如otherCollectionMethods.wxml等）。

5.7.3　修改文件otherCollectionMethods.wxml

修改文件otherCollectionMethods.wxml，文件otherCollectionMethods.wxml修改后的代码如例5-12所示。

【例5-12】 文件otherCollectionMethods.wxml修改后的代码示例。

```
<!-- pages/otherCollectionMethods/otherCollectionMethods.wxml -->
<text>pages/otherCollectionMethods/otherCollectionMethods.wxml</text>
<button type="primary" bindtap="countMethod">count 方法</button>
<button type="primary" bindtap="countPromise">Promise 方式 count 方法</button>
<button type="primary" bindtap="orderByPrice">按 Price 排序</button>
<button type="primary" bindtap="orderByMulti">组合排序</button>
<button type="primary" bindtap="fieldMethod">field 方法</button>
<button type="primary" bindtap="watchQueryEx">watch 查询条件</button>
<button type="primary" bindtap="watchDocEx">watch 记录</button>
```

5.7.4　修改文件otherCollectionMethods.js

修改文件otherCollectionMethods.js，文件otherCollectionMethods.js修改后的代码如例5-13所示。

【例5-13】 文件otherCollectionMethods.js修改后的代码示例。

```
//pages/otherCollectionMethods/otherCollectionMethods.js
Page({
  countMethod: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
      _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI'
    }).count({
      success: function(res) {
        console.log("记录的总数是: " + res.total)
      }
```

```
  })
},
countPromise: function() {
  const db = wx.cloud.database()
  db.collection('mpcloudbook').where({
    _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI'
  }).count().then(res => {
    console.log(res.total)
  })
},
orderByPrice: function() {
  const db = wx.cloud.database()
  db.collection('mpcloudbook').orderBy('price', 'asc')
    .get({
      success: function(res) {
        console.log(res.data)
      }
    })
},
orderByMulti: function() {
  const db = wx.cloud.database()
  db.collection('mpcloudbook')
    .orderBy('price', 'desc')
    .orderBy('description', 'asc')
    .get({
      success: function(res) {
        console.log(res.data)
      }
    })
},
fieldMethod: function() {
  const db = wx.cloud.database()
  db.collection('mpcloudbook').field({
      description: true,
      id: true,
      price: true,
      //只返回 topics 数组前 2 个元素
      topics: true,
    })
    .get({
      success: function(res) {
        console.log(res.data)
      }
    })
},
```

```
  watchQueryEx: function() {
    const db = wx.cloud.database()
    const watcher = db.collection('mpcloudbook').where({
      _openid: 'oHmb80Bf7vaqwlyQaLTCfOlgOVlI'
    }).watch({
      onChange: function(snapshot) {
        console.log('snapshot', snapshot)
      },
      onError: function(err) {
        console.error('the watch closed because of error', err)
      }
    })
  },
  watchDocEx: function() {
    const db = wx.cloud.database()
    const watcher = db.collection('mpcloudbook').doc('Spring1').watch({
      onChange: function(snapshot) {
        console.log('snapshot', snapshot)
      },
      onError: function(err) {
        console.error('the watch closed because of error', err)
      }
    })
  }
})
```

5.7.5　运行程序

编译程序，模拟器中的输出结果如图 5-12 所示。从顶部向底部依次单击图 5-12 中的 7 个按钮，控制台中的输出结果如图 5-13 所示。

图 5-12　编译程序后模拟器中的输出结果

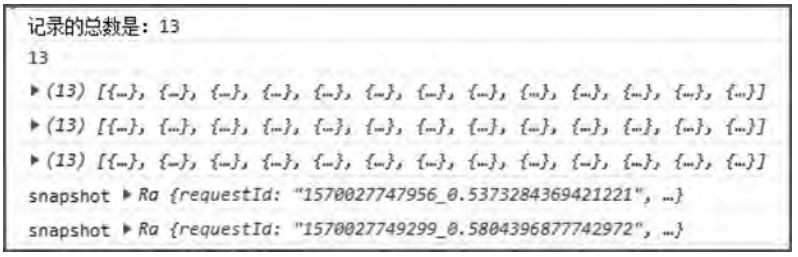

图 5-13　从顶部向底部依次单击图 5-12 中 7 个按钮后控制台中的输出结果

5.8　在小程序端正则表达式的用法

视频讲解

5.8.1　API 说明

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() 方法构造正则对象后进行字符串匹配。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)。

使用正则表达式匹配可以满足字符串匹配需求，但不适用于长文本、大数据量的文本匹配、搜索，因为这样操作会有性能问题，对此类操作应该使用文本搜索引擎(如 ElasticSearch 等)实现。

db.RegExp()的参数 options 支持 i、m、s 共 3 个 flag，而 JavaScript 原生正则对象构造时仅支持其中的 i、m 两个 flag，因此当需要使用 s 这个 flag 时必须使用 db.RegExp 构造器构造正则对象。flag 的信息如表 5-9 所示。

表 5-9　flag 的信息

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

5.8.2　辅助工作

在 5.6 节项目 secondcloud 和数据库集合 mpcloudbook 的基础上继续后续的开发。

修改文件 app.json，代码的修改方法是在语句“"pages/otherCollectionMethods/otherCollectionMethods",”之前增加语句“"pages/dbRegExp/dbRegExp",”。修改代码后编译程序，自动在目录 pages 下生成 dbRegExp 子目录，且在 pages/dbRegExp 目录下自动生成了 dbRegExp 页面的 4 个文件(如 dbRegExp.wxml 等)。

5.8.3　修改文件 dbRegExp.wxml

修改文件 dbRegExp.wxml，文件 dbRegExp.wxml 修改后的代码如例 5-14 所示。

【例 5-14】　文件 dbRegExp.wxml 修改后的代码示例。

```
<!-- pages/dbRegExp/dbRegExp.wxml -->
<text>pages/dbRegExp/dbRegExp.wxml</text>
<button type="primary" bindtap="nativeJSObject">原生 JavaScript 对象</button>
<button type="primary" bindtap="dbREObject">数据库正则对象</button>
<button type="primary" bindtap="newConstructor">new 构造方法</button>
```

5.8.4　修改文件 dbRegExp.js

修改文件 dbRegExp.js，文件 dbRegExp.js 修改后的代码如例 5-15 所示。

【例 5-15】　文件 dbRegExp.js 修改后的代码示例。

```
//pages/dbRegExp/dbRegExp.js
Page({
  nativeJSObject: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
      description: /miniprogram/i
    }).get({
      success: function(res) {
        console.log(res.data)
      }
    })
  },
  dbREObject: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
      description: db.RegExp({
        regexp: 'spring',
        options: 'i',
      })
    }).get({
      success: function(res) {
        console.log(res.data)
      }
    })
  },
  newConstructor: function() {
    const db = wx.cloud.database()
    db.collection('mpcloudbook').where({
      _id: new db.RegExp({
```

```
          regexp: 'miniprogram',
          options: 'i',
        })
      }).get({
        success: function(res) {
          console.log(res.data)
        }
      })
    }
  })
```

5.8.5 运行程序

编译程序,模拟器中的输出结果如图 5-14 所示。从顶部向底部依次单击图 5-14 中的 3 个按钮,控制台中的输出结果如图 5-15 所示。

图 5-14 编译程序后模拟器中的输出结果

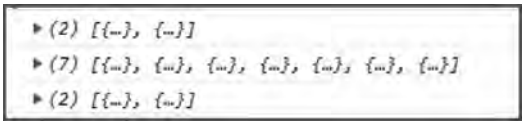

图 5-15 从顶部向底部依次单击图 5-14 中 3 个按钮后控制台中的输出结果

5.9 在小程序端处理地理信息 db.Geo

视频讲解

5.9.1 API 说明

db.Geo 对象上含有地理位置构造器。为了使用基于地理位置的查询,必须为相应存放地理位置的地方添加地理位置索引。

在使用地理位置接口时,除了使用下述提供的各种地理位置构造器外,均可使用其等价的 GeoJSON 表示法。通过地理位置构造器构造的各个对象均可调用方法 toJSON()获得其等价的 GeoJSON 纯 JavaScript 对象。在查询结果中如果含有地理位置字段,则均返回地理位置对象而不是 GeoJSON 对象。

db.Geo 拥有 Point()方法、LineString()方法、Polygon()方法、MultiPoint()方法、MultiLineString()方法、MultiPolygon()方法。

db.Geo.Point()方法构造一个点，方法接收两个必填参数：第一个是经度(longitude)；第二个是纬度(latitude)。务必注意两个参数的顺序。db.Geo.MultiPoint()方法构造一个点集合，点集合由一个或更多的点组成。

db.Geo.LineString()方法构造一条线段，线段由两个或更多的点有序连接组成。db.Geo.MultiLineString()方法构造一个线段，线段集合由多条线段组成。

db.Geo.Polygon()方法构造一个多边形，多边形由一个或多个线性环(Linear Ring)组成，线性环即一个闭合的线段。一个闭合线段至少由4个点组成，其中最后一个点和第一个点的坐标必须相同，以此表示环的起点和终点。如果一个多边形由多个线性环组成，则第一个线性环表示外环(外边界)，接下来的所有线性环表示内环(即外环中的洞，不计在此多边形中的区域)。如果一个多边形只有一个线性环组成，则这个环就是外环。多边形构造规则：第一个线性环必须是外环，外环不能自交，所有内环必须完全在外环内，各个内环间不能相交或重叠，也不能有共同的边。db.Geo.MultiPolygon()方法构造一个多边形集合，多边形集合由多个多边形组成。

5.9.2　辅助工作

按照3.2.1节的方法，在环境 learnwxbookscode(环境 ID 为 learnwxbookscode-wsd001)中创建一个数据库集合 activities。

在5.8节项目 secondcloud 和数据库集合 activities 的基础上继续后续的开发。

修改文件 app.json，代码的修改方法是在语句“"pages/dbRegExp/dbRegExp",”之前增加语句“"pages/dbGeoEx/dbGeoEx",”。修改代码后编译程序，自动在目录 pages 下生成 dbGeoEx 子目录，且在 pages/dbGeoEx 目录下自动生成了 dbGeoEx 页面的4个文件(如 dbGeoEx.wxml 等)。

5.9.3　修改文件 dbGeoEx.wxml

修改文件 dbGeoEx.wxml，文件 dbGeoEx.wxml 修改后的代码如例5-16所示。

【例5-16】 文件 dbGeoEx.wxml 修改后的代码示例。

```
<!-- pages/dbGeoEx/dbGeoEx.wxml -->
<button type="primary" bindtap="pointEx">点</button>
<button type="primary" bindtap="pointJSONEx">JSON 表示点</button>
<button type="primary" bindtap="lineEx">线段</button>
<button type="primary" bindtap="lineJSONEx">JSON 表示线段</button>
<button type="primary" bindtap="polygonEx">单环多边形</button>
<button type="primary" bindtap="multipolygonEx">多环多边形</button>
<button type="primary" bindtap="polygonJSONEx">JSON 表示多边形</button>
<button type="primary" bindtap="pointsJSONEx">JSON 表示点的集合</button>
<button type="primary" bindtap="linesJSONEx">JSON 表示线段集合</button>
```

```
<button type = "primary" bindtap = "polygonsEx">多边形集合</button>
<button type = "primary" bindtap = "polygonsJSONEx">JSON 表示多边形集合</button>
```

5.9.4 修改文件 dbGeoEx.js

修改文件 dbGeoEx.js,文件 dbGeoEx.js 修改后的代码如例 5-17 所示。

【例 5-17】 文件 dbGeoEx.js 修改后的代码示例。

```
//pages/dbGeoEx/dbGeoEx.js
Page({
  pointEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Get up',
        location: db.Geo.Point(113, 23)
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  pointJSONEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Have breakfast',
        location: {
          type: 'Point',
          coordinates: [113, 23]
        }
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  lineEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Go to school',
        location: db.Geo.LineString([
          db.Geo.Point(113, 23),
          db.Geo.Point(120, 50),
```

```
          //... 可选更多点
        ])
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  lineJSONEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Go to class',
        location: {
          type: 'LineString',
          coordinates: [
            [113, 23],
            [120, 50]
          ]
        }
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  polygonEx: function() {
    const db = wx.cloud.database()
    const {
      Polygon,
      LineString,
      Point
    } = db.Geo
    db.collection('activities').add({
      data: {
        description: 'Recess',
        location: Polygon([
          LineString([
            Point(0, 0),
            Point(3, 2),
            Point(2, 3),
            Point(0, 0)
          ])
        ])
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
```

```
      fail: console.error
    })
  },
  multipolygonEx: function() {
    const db = wx.cloud.database()
    const {
      Polygon,
      LineString,
      Point
    } = db.Geo
    db.collection('activities').add({
      data: {
        description: 'Have lunch',
        location: Polygon([
          //外环
          LineString([Point(0, 0), Point(30, 20), Point(20, 30), Point(0, 0)]),
          //内环
          LineString([Point(10, 10), Point(16, 14), Point(14, 16), Point(10, 10)])
        ])
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  polygonJSONEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Go home',
        location: {
          type: 'Polygon',
          coordinates: [
            [
              [0, 0],
              [30, 20],
              [20, 30],
              [0, 0]
            ],
            [
              [10, 10],
              [16, 14],
              [14, 16],
              [10, 10]
            ]
          ]
        }
      },
      success: function(res) {
```

```
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  pointsJSONEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Have dinner',
        location: {
          type: 'MultiPoint',
          coordinates: [
            [113, 23],
            [120, 50]
          ]
        }
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  linesJSONEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Take exercise',
        location: {
          type: 'MultiLineString',
          coordinates: [
            [
              [0, 0],
              [3, 3]
            ],
            [
              [5, 10],
              [20, 30]
            ]
          ]
        }
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  polygonsEx: function() {
```

```
    const db = wx.cloud.database()
    const {
      MultiPolygon,
      Polygon,
      LineString,
      Point
    } = db.Geo
    db.collection('activities').add({
      data: {
        description: 'Do homework',
        location: MultiPolygon([
          Polygon([
            LineString([Point(50, 50), Point(60, 80), Point(80, 60), Point(50, 50)]),
          ]),
          Polygon([
            LineString([Point(0, 0), Point(30, 20), Point(20, 30), Point(0, 0)]),
            LineString([Point(10, 10), Point(16, 14), Point(14, 16), Point(10, 10)])
          ]),
        ])
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  },
  polygonsJSONEx: function() {
    const db = wx.cloud.database()
    db.collection('activities').add({
      data: {
        description: 'Go to bed',
        location: {
          type: 'MultiPolygon',
          coordinates: [
            [
              [
                [50, 50],
                [60, 80],
                [80, 60],
                [50, 50]
              ]
            ],
            [
              [
                [0, 0],
                [30, 20],
```

```
                [20, 30],
                [0, 0]
              ],
              [
                [10, 10],
                [16, 14],
                [14, 16],
                [10, 10]
              ]
            ]
          ]
        }
      },
      success: function(res) {
        console.log("成功添加记录: " + res._id)
      },
      fail: console.error
    })
  }
})
```

5.9.5　运行程序

编译程序,模拟器中的输出结果如图 5-16 所示。从顶部向底部依次单击图 5-16 中的 11 个按钮,控制台中的输出结果如图 5-17 所示。

图 5-16　编译程序后模拟器中的输出结果

```
成功添加记录: 1af3506e5d955250097fe4b321fc0d98
成功添加记录: 1af3506e5d955252097fe59c6d2575da
成功添加记录: 075734515d955253097fb79b13523007
成功添加记录: f885cb355d955255097f90d871ab45fc
成功添加记录: 075734515d955256097fb9d2447f0cca
成功添加记录: 3397e9015d955258097f1f5c785ae197
成功添加记录: 1af3506e5d95525a097feb372da90b06
成功添加记录: 075734515d95525c097fbde102207a99
成功添加记录: f885cb355d95525e097f971b4055196d
成功添加记录: 075734515d95525f097fbff56380e260
成功添加记录: 075734515d955261097fc14d638127f6
```

图 5-17　从顶部向底部依次单击图 5-16 中 11 个按钮后控制台中的输出结果

5.10　在小程序端聚合的用法

视频讲解

5.10.1　聚合说明

聚合是一个流水线式的批处理作业(操作),包含多个批处理阶段。每个阶段接收来自上一个阶段的输入记录列表(如果是第一个阶段则是集合全集),然后将数据分组(或者不分组,不分组时只有一组,每条记录都是一组),对每组数据执行多种批处理操作,处理成新的记录列表后输出给下一个阶段,直至返回结果。

聚合可以应用于:

(1) 分组查询,例如,按图书类别获取各类图书的平均销量。

(2) 只取某些字段的统计值或变换值返回,例如,当每条图书记录中存放了一个数组字段代表每月销量时想要获取图书的月平均销量。

(3) 流水线式分阶段批处理,例如,求各图书类别的总销量最大的作者和最小的作者的操作。

(4) 获取唯一值,例如,获取某个类别的图书的所有作者名时需去掉重复值。

一个聚合阶段是将一批输入记录按开发者指定的规则转换为新一批输出记录的过程。一个阶段的输出记录数与其输入记录数无关,既可以保持不变,每个输入记录对应一个输出记录,也可以合并或分组输出更少的一个或多条记录,甚至于输出更多的记录。一个聚合流水线操作的第一个阶段是流水线的开始,接收集合的所有记录作为输入,最后一个阶段是流水线的结束,其结果作为输出返回给调用方。要定义一个阶段,首先确定要使用的阶段,聚合功能提供了包括分组阶段 group、排序阶段 sort、投影阶段 project 等多种可选的阶段。每个阶段又可以通过一个对象作为参数定义这个阶段操作的具体行为表现,其中该参数对象的每个字段的值都必须是一个表达式或聚合操作符,一个操作符可以接收表达式作为输入(常量、字段引用等)。

在聚合中,一个表达式可以是字段(路径)引用、常量、对象表达式或操作符表达式,并且可以嵌套使用表达式。通过字段(路径)引用可以引用一个字段的值,以一个 $ 开头的字符串代表字段(路径)引用,例如 $exam 表示引用 exam 字段,如果是嵌套字段或数组,也可以

通过点表示法和数组下标表示法取引用，例如 $exam.math 表示引用 exam 字段对象下的 math 字段，$score[0]表示引用数组字段 score 的第一个元素。表达式还可以是数字、字符串等常量，如果要使用一个以 $ 开头的字符串常量，需要使用 $literal 表示这是一个常量而不是字段引用。对象表达式即一个每个字段的值都是一个表达式的对象。

5.10.2　API 说明

聚合 API 包括所有聚合流水线阶段、聚合操作符、发起和结束调用的接口。

聚合有 aggregate、end 两个基本阶段。aggregate 阶段添加新字段到输出的记录。经过 addFields 聚合阶段，输出的所有记录中除了输入时带有的字段外，还将带有 addFields 指定的字段。end 阶段将输入记录根据给定的条件和边界划分成不同的组，每组即一个 bucket（桶）。

桶是对象存储服务（Object Storage Service，OSS 或简称 OBS）中存储对象的容器。OBS 是一个基于对象的海量存储服务，为客户提供海量、安全、高可靠、低成本的数据存储功能，包括创建、修改、删除桶，上传、下载、删除对象等。对象存储提供了基于桶和对象的扁平化存储方式，桶中的所有对象都处于同一逻辑层级，去掉了文件系统中的多层级树形目录结构。每个桶都有自己的存储类别、访问权限、所属区域等属性，用户可以在不同区域创建不同存储类别和访问权限的桶，并配置更多高级属性来满足不同场景的存储需求。

OBS 系统和单个桶都没有总数据容量和对象/文件数量的限制，为用户提供了超大存储容量的功能，适合存放任意类型的文件，适合普通用户、网站、企业和开发者使用。由于 OBS 是基于 REST 风格 HTTP 和 HTTPS 的服务，可以通过 URL（Uniform Resource Locator，统一资源定位符）定位资源。OBS 提供了基于 HTTP/HTTPS 的 Web 服务接口，用户可以随时随地访问可连接到 Internet 的计算机，通过 OBS 管理控制台或客户端访问和管理存储在 OBS 中的数据。此外，OBS 支持 SDK（Software Development Kit，软件开发工具包）和 API，可使用户方便地管理自己存储在 OBS 上的数据，以及开发多种类型的上层业务应用。OBS 可供用户存储任意类型和大小的数据，适合企业备份/归档、视频点播、视频监控等多种数据存储场景。OBS 还提供图片处理特性，为用户提供稳定、安全、高效、易用、低成本的图片处理服务，包括图片剪切、图片缩放、图片水印、格式转换等。

聚合流水线阶段信息如表 5-10 所示。

表 5-10　聚合流水线阶段信息

阶　段	描　述
addFields	添加新字段到输出的记录。经过 addFields 聚合阶段，输出的所有记录中除了输入时带有的字段外，还将带有 addFields 指定的字段
bucket	将输入记录根据给定的条件和边界划分成不同的组，每组即一个 bucket
bucketAuto	将输入记录根据给定的条件划分成不同的组，每组即一个 bucket 与 bucket 的其中一个不同之处在于无须指定 boundaries，bucketAuto 会自动尝试将记录尽可能平均地分散到每组中
count	计算输入记录数，输出一条记录，其中指定字段的值为记录数
geoNear	将记录按照离给定点从近到远输出

续表

阶　段	描　述
group	将输入记录按给定表达式分组，输出时每条记录代表一个分组，每条记录的 _id 是区分不同组的 key。输出记录中也可以包括累计值，将输出字段设为累计值即会从该分组中计算累计值
limit	限制输出到下一阶段的记录数
match	根据条件过滤文档，并且把符合条件的文档传递给下一个流水线阶段
project	把指定的字段传递给下一个流水线，指定的字段可以是某个已经存在的字段，也可以是计算出来的新字段
replaceRoot	指定一个已有字段作为输出的根节点，也可以指定一个计算出的新字段作为根节点
sample	随机从文档中选取指定数量的记录
skip	指定一个正整数，跳过对应数量的文档，输出剩下的文档
sort	根据指定的字段，对输入的文档进行排序
sortByCount	根据传入的表达式，将传入的集合进行分组。然后计算不同组的数量，并且将这些组按照它们的数量进行排序，返回排序后的结果
unwind	使用指定的数组字段中的每个元素，对文档进行拆分。拆分后，文档会从一个变为一个或多个，分别对应数组的每个元素

因为每个聚合操作是要输入整个集合的数据的，因此为了优化执行效率聚合有以下基本使用原则。

（1）match 和 sort 如果是在流水线的开头则可以利用索引。geoNear 也可以利用地理位置索引，但要注意的是，geoNear 必须是流水线的第一个阶段。match 参数的语法与普通查询语法相同。除了 match 阶段，在各个聚合阶段中传入的对象可使用的操作符都是聚合操作符，需要特别注意的是，match 进行的是查询匹配，因此语法同普通查询（where）的语法，用的是普通查询操作符。

（2）只要需要的不是集合的全集，那就应该尽早地通过 match、limit 和 skip 缩小要处理的记录数量。

聚合操作符信息如表 5-11 所示。

表 5-11　聚合操作符信息

操　作　符	描　述
abs	返回一个数字的绝对值
add	将数字相加或将数字加在日期上。如果数组中的其中一个值是日期，那么其他值将被视为毫秒数加在该日期上
addToSet	向数组中添加值，如果数组中已存在该值，不执行任何操作。它只能在分组阶段中使用
allElementsTrue	输入一个数组，或者数组字段的表达式。如果数组中所有元素均为真值，那么返回 true，否则返回 false。空数组永远返回 true
and	给定多个表达式，and 仅在所有表达式都返回 true 时返回 true，否则返回 false
anyElementTrue	输入一个数组，或者数组字段的表达式。如果数组中任意一个元素为真值，那么返回 true，否则返回 false。空数组永远返回 false
arrayElemAt	返回在指定数组下标的元素
arrayToObject	将一个数组转换为对象

续表

操　作　符	描　　述
avg	返回一组集合中，指定字段对应数据的平均值
ceil	向上取整，返回大于或等于给定数字的最小整数
cmp	给定两个值，返回其比较值
concat	连接字符串，返回拼接后的字符串
concatArrays	将多个数组拼接成一个数组
cond	计算布尔表达式，返回指定的两个值中的一个
dateFromParts	给定日期的相关信息，构建并返回一个日期对象
dateFromString	将一个日期/时间字符串转换为日期对象
dateToString	根据指定的表达式将日期对象格式化为符合要求的字符串
dayOfMonth	返回日期字段对应的天数(一个月中哪一天)，是一个 1～31 的数字
dayOfWeek	返回日期字段对应的天数(一周中第几天)，是一个 1(周日)～7(周六)的整数
dayOfYear	返回日期字段对应的天数(一年中第几天)，是一个 1～366 的整数
divide	传入被除数和除数，求商
eq	匹配两个值，如果相等则返回 true，否则返回 false
exp	取 e(自然对数的底数，欧拉数)的 n 次方
filter	根据给定条件返回满足条件的数组的子集
first	返回指定字段在一组集合的第一条记录对应的值。仅当这组集合是按照某种定义排序后，此操作才有意义
floor	向下取整，返回大于或等于给定数字的最小整数
gt	匹配两个值，如果前者大于后者则返回 true，否则返回 false
gte	匹配两个值，如果前者大于或等于后者则返回 true，否则返回 false
hour	返回日期字段对应的小时数，是一个 0～23 的整数
ifNull	计算给定的表达式，如果表达式结果为 null、undefined 或者不存在，那么返回一个替代值；否则返回原值
in	给定一个值和一个数组，如果值在数组中则返回 true，否则返回 false
indexOfArray	在数组中找出等于给定值的第一个元素的下标，如果找不到则返回－1
indexOfBytes	在目标字符串中查找子字符串，并返回第一次出现的 UTF-8 的字节索引(从 0 开始)。如果不存在子字符串，则返回－1
indexOfCP	在目标字符串中查找子字符串，并返回第一次出现的 UTF-8 的 codepoint 索引(从 0 开始)。如果不存在子字符串，则返回－1
isArray	判断给定表达式是否是数组，返回布尔值
isoDayOfWeek	返回日期字段对应的 ISO 8601 标准的天数(一周中第几天)，是一个 1(周一)～7(周日)的整数
isoWeek	返回日期字段对应的 ISO 8601 标准的周数(一年中第几周)，是一个 1～53 的整数
isoWeekYear	返回日期字段对应的 ISO 8601 标准的天数(一年中的第几天)
last	返回指定字段在一组集合的最后一条记录对应的值。仅当这组集合是按照某种定义排序后，此操作才有意义
let	自定义变量，并且在指定表达式中使用，返回的结果是表达式的结果
literal	直接返回一个值的字面量，不经过任何解析和处理
ln	计算给定数字在自然对数值
log	计算给定数字在给定对数底下的 log 值

续表

操 作 符	描 述
log10	计算给定数字在对数底为 10 下的 log 值
lt	匹配两个值，如果前者小于后者则返回 true，否则返回 false
lte	匹配两个值，如果前者小于或等于后者则返回 true，否则返回 false
map	类似于 JavaScript 中数组类型上的 map 方法，将给定数组的每个元素按给定转换方法转换后得出新的数组
max	返回一组数值的最大值
mergeObjects	将多个文档合并为单个文档
millisecond	返回日期字段对应的毫秒数，是一个 0～999 的整数
min	返回一组数值的最小值
minute	返回日期字段对应的分钟数，是一个 0～59 的整数
mod	取模运算，取数字取模后的值
month	返回日期字段对应的月份，是一个 1～12 的整数
multiply	取传入的数字参数相乘的结果
neq	匹配两个值，如果不相等则返回 true，否则返回 false
not	给定一个表达式，如果表达式返回 true，则 not 返回 false，否则返回 true
objectToArray	将一个对象转换为数组。方法把对象的每个键值对都变成输出数组的一个元素，元素形如{k：<key>，v：<value>}
or	给定多个表达式，如果任意一个表达式返回 true，则 or 返回 true，否则返回 false
pow	求给定基数的指数次幂
push	在 group 阶段，返回一组中表达式指定列与对应的值一起组成的数组
range	返回一组生成的序列数字。给定开始值、结束值、非零的步长，range 会返回从开始值开始逐步增长、步长为给定步长但不包括结束值的序列
reduce	类似于 JavaScript 中 reduce 方法，将一个表达式应用于数组中各个元素然后归一成一个元素
reverseArray	返回给定数组的倒序形式
second	返回日期字段对应的秒数，是一个 0～59 的整数，在特殊情况下（闰秒）可能等于 60
setDifference	输入两个集合，输出只存在于第一个集合中的元素
setEquals	输入两个集合，判断两个集合中包含的元素是否相同（不考虑顺序、去重）
setIntersection	输入两个集合，输出两个集合的交集
setIsSubset	输入两个集合，判断第一个集合是否是第二个集合的子集
setUnion	输入两个集合，输出两个集合的并集
size	返回数组长度
slice	类似于 JavaScript 中 slice()方法，返回给定数组的指定子集
split	按照分隔符分隔数组，并且删除分隔符，返回子字符串组成的数组。如果字符串无法找到分隔符进行分隔，则返回原字符串作为数组的唯一元素
sqrt	求平方根
stdDevPop	返回一组字段对应值的标准差
stdDevSamp	计算输入值的样本标准偏差。如果输入值代表数据总体或者不概括更多的数据，则改用 db.command.aggregate.stdDevPop
strLenBytes	计算并返回指定字符串中 UTF-8 编码的字节数量
strLenCP	计算并返回指定字符串的 UTF-8 代码点数量

续表

操　作　符	描　　述
strcasecmp	对两个字符串在不区分大小写的情况下进行大小比较，并返回比较的结果
substr	返回字符串从指定位置开始的指定长度的子字符串。它是 db.command.aggregate.substrBytes 的别名，推荐使用后者
substrBytes	返回字符串从指定位置开始的指定长度的子字符串。子字符串是由字符串中指定的 UTF-8 字节索引的字符开始，长度为指定的字节数
substrCP	返回字符串从指定位置开始的指定长度的子字符串。子字符串是由字符串中指定的 UTF-8 字节索引的字符开始，长度为代码点数
subtract	将两个数字相减然后返回差值，或将两个日期相减然后返回相差的毫秒数，或将一个日期减去一个数字返回结果的日期
sum	计算并且返回一组字段所有数值的总和
switch	根据给定的 switch-case-default 计算返回值
toLower	将字符串转换为小写并返回
toUpper	将字符串转换为大写并返回
trunc	将数字截断为整型
week	返回日期字段对应的周数(一年中的第几周)，是一个 1～53 的整数
year	返回日期字段对应的年份
zip	把二维数组的第二维数组中的相同序号的元素分别拼接成一个新的数组进而组成一个新的二维数组。如可将[[1,2,3],["a","b","c"]]转换成[[1,"a"],[2,"b"],[3,"c"]]

5.10.3　辅助工作

按照 3.2.1 节的方法，在环境 learnwxbookscode(环境 ID 为 learnwxbookscode-wsd001)中创建一个数据库集合 aggdb。

在 5.8 节项目 secondcloud 和数据库集合 aggdb 的基础上继续后续的开发。

修改文件 app.json，代码的修改方法是在语句“"pages/dbGeoEx/dbGeoEx",”之前增加语句“"pages/dbAggEx/dbAggEx",”。修改代码后编译程序，自动在目录 pages 下生成 dbAggEx 子目录，且在 pages/dbAggEx 目录下自动生成了 dbAggEx 页面的 4 个文件(如 dbAggEx.wxml 等)。

5.10.4　修改文件 dbAggEx.wxml

修改文件 dbAggEx.wxml，文件 dbAggEx.wxml 修改后的代码如例 5-18 所示。

【例 5-18】 文件 dbAggEx.wxml 修改后的代码示例。

```
<!-- pages/dbAggEx/dbAggEx.wxml -->
<text>pages/dbAggEx/dbAggEx.wxml</text>
<button type="primary" bindtap="simpleex">简单运算的结果</button>
<button type="primary" bindtap="arrayex">数组运算的结果</button>
<button type="primary" bindtap="stringex">字符串运算的结果</button>
```

```
<button type = "primary" bindtap = "objectex">对象运算的结果</button>
<button type = "primary" bindtap = "addFieldsex"> addFields 阶段</button>
<button type = "primary" bindtap = "replaceRootex"> replaceRoot 阶段</button>
<button type = "primary" bindtap = "sampleex"> sample 阶段</button>
```

5.10.5 修改文件 dbAggEx.js

修改文件 dbAggEx.js,文件 dbAggEx.js 修改后的代码如例 5-19 所示。

【例 5-19】 文件 dbAggEx.js 修改后的代码示例。

```
//pages/dbAggEx/dbAggEx.js
Page({
  simpleex: function() {
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .project({
        absresult: $.abs($.subtract(['$begin', '$end'])),
        total: $.add(['$begin', '$end']),
        average: $.avg('$price'),
        salesprice: $.ceil('$price'),
        compare: $.cmp(['$begin', '$end']),
        discount: $.cond({
          if: $.gte(['$price', 50]),
          then: 0.7,
          else: 0.9
        }),
        constructdate: $.dateFromParts({
          year: 2017,
          month: 2,
          day: 8,
          hour: 12,
          timezone: 'America/New_York'
        }),
        transferdate: $.dateFromString({
          dateString: "2019-05-14T09:38:51.686Z"
        }),
        km: $.divide(['$begin', 1000]),
        expdelta: $.exp('$delta'),
        orderprice: $.floor('$price'),
        expensive: $.gte(['$price', 50]),
        sqrtresult: $.sqrt([$.add([$.pow(['$begin', 2]), $.pow(['$end', 2])])]),
        points: $.range([0, '$end', 200]),
        fullfilled: $.or([$.lt(['$delta', 5]), $.gt(['$begin', 60])]),
        isAllTrue: $.allElementsTrue(['$tags']),
        dayOfMonth: $.dayOfMonth('$date'),
        dayOfWeek: $.dayOfWeek('$date'),
```

```
        dayOfYear: $.dayOfYear('$date'),
        description: $.ifNull(['$description', '描述空缺']),
        index: $.indexOfArray(['$tags', 2, 2])
      }).end().then(console.log)
  },
  arrayex: function() {
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .group({
        _id: 'aggdbcategory',
        newcategories: $.addToSet('$category'),
        tagsList: $.addToSet('$tags'),
      })
      .end().then(console.log)
  },
  stringex: function() {
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .project({
        _id: 'string ex result',
        itskills: $.concat(['$description', '', '$category']),
        comercelist: $.concatArrays(['$commerces', '$tags']),
        aStrIndex: $.indexOfBytes(['$description', 'a']),
        foobar: $.mergeObjects(['$foo', '$bar']),
        result: $.toLower('$description')
      })
      .end().then(console.log)
  },
  objectex: function() {
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .project({
        foobar: $.mergeObjects(['$foo', '$bar']),
        arrayfoo: $.objectToArray('$foo')
      })
      .end().then(console.log)
  },
  addFieldsex: function() {
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .addFields({
        totalres: $.add(['$begin', '$delta', '$end'])
      })
      .end().then(console.log)
  },
  replaceRootex: function() {
```

```
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .replaceRoot({
        newRoot: '$foo'
      })
      .end().then(console.log)
  },
  sampleex: function() {
    const db = wx.cloud.database()
    const $ = db.command.aggregate
    db.collection('aggdb').aggregate()
      .sample({
        size: 1
      })
      .end().then(console.log)
  }
})
```

5.10.6 运行程序

编译程序,模拟器中的输出结果如图 5-18 所示。从顶部向底部依次单击图 5-18 中的 7 个按钮,控制台中的输出结果如图 5-19 所示。

图 5-18 编译程序后模拟器中的输出结果

```
▶ {list: Array(2), errMsg: "collection.aggregate:ok"}
▶ {list: Array(1), errMsg: "collection.aggregate:ok"}
▶ {list: Array(2), errMsg: "collection.aggregate:ok"}
▶ {list: Array(2), errMsg: "collection.aggregate:ok"}
▶ {list: Array(2), errMsg: "collection.aggregate:ok"}
▶ {list: Array(2), errMsg: "collection.aggregate:ok"}
▶ {list: Array(1), errMsg: "collection.aggregate:ok"}
```

图 5-19 从顶部向底部依次单击图 5-18 中 7 个按钮后控制台中的输出结果

习题 5

简答题

1. 简述对数据库数据类型的理解。
2. 简述对数据库权限控制的理解。
3. 简述对数据库初始化的理解。

实验题

1. 实现在小程序端往集合中插入数据。
2. 实现在小程序端查询数据。
3. 实现小程序端查询指令的应用。
4. 实现在小程序端更新数据。
5. 实现小程序端更新指令的应用。
6. 实现在小程序端删除数据。
7. 实现小程序端对集合其他操作方法的应用。
8. 实现在小程序端正则表达式的应用。
9. 实现在小程序端处理地理信息 db.Geo 的应用。
10. 实现小程序端聚合的应用。