数据库操作基本可以看文档解决，（主要是按加号新增数据等操作要了解）

连接数据库（js中的最外层，page外）

const db = wx.cloud.database()

指定数据库（在js，方法中）

db.collection("数据库名字")

查询

.get()方法就是查询

db.collection("demolist").get({

      success : res=>{

        console.log(res)

      }

    })

若在.get方法前加入一个.doc(“\_id”)，则可以筛选数据（只能放id）

db.collection("demolist").get().then(res=>{

      console.log(res)

      })

这种方法也是可以的，这里then就是相当于success的用法。返回结果完全一致。

如果有调用嵌套，继续在后面加then即可。

    db.collection("demolist").get().then(res=>{

      console.log(res)

      }).catch(err=>{

      })

但通常我们还会加入catch来捕获错误。

若知道其他数据内容（如知道名字，但不知道id），就可以用where

    db.collection("demolist").where({

      author:"作者"

    }).get()

    .then(res=>{

      console.log(res)

    })

（大于小于之类的操作，等sdk再详细讲解）

添加

.add()添加数据

    db.collection("demolist").add({

      data:{

        title : "测试标题2",

        author : "李四",

        content : "测试内容2"

      }

    }).then(res=>{

      console.log(res)

    })

这里的res（then）可以不写，不过最好写一下，查询有没有错误

添加记录的时候，会自动写入openid（每一个人的openid是一样的）

    wx.showLoading({

      title: '数据加载中',

      mask:true

    })

    //这一步用于放止用户一直点击

    db.collection("demolist").add({

      data:{

        title : "测试标题2",

        author : "李四",

        content : "测试内容2"

      }

    }).then(res=>{

      console.log(res)

      wx.hideLoading()

    })

在最前面加一个showloading和后面用hideloading，放止用户一直点击。

表单提交（用输入框提交到数据库里）

  btnSub(res){

    var title = res.detail.value.title;

    var author = res.detail.value.author;

    var content = res.detail.value.content;

    db.collection("demolist").add({

      data : {

        title : title,

        author :author,

        content : content

      }

    }).then(res=>{

      console.log(res)

    })

  }

安全写法。

var resVlu = res.detail.value;

    db.collection("demolist").add({

      data:resVlu

    }).then(res=>{

      console.log(res)

    })

简洁写法。

data:resVlu

就可以自动的做类似于上面content:content的操作。

系统会自动写入openid，我们可以通过用户的openid来查询数据

更新数据

    db.collection("demolist").doc("f2a60d815edf897a00727b77257d4935").update({

      data : {

        author:"作者"

      }

    }).then(res=>{

      console.log(res)

    })

这里也可以用where，不用doc。Doc就只能改一条数据，where可以改多个。

（where要用云函数）

更新操作不仅可以修改数据，还能增加数据，

如果用的不是update而是用set方法，就会把其他字段覆盖。比如只写了author，其他数据都会消失。

删除

    db.collection("demolist").doc("").remove().then(res=>{

      console.log(res)

    })

Where操作也是要云函数

.count()统计个数

.watch()监听

通常写在onLoad中，比如有一条记录被删除，他就能自动重新调用监听方法，看到哪条记录被删除，剩下哪条记录。

    db.collection("demolist").watch({

      onChange:function(res){

        console.log(res)

      },

      onError:function(err){

        console.log(err)

      }

    })

onChange和onError方法是必须调用。



docChanges是改变了哪一条，docs是现在剩下的情况。



用这种方法就可以把数据渲染到前端去

  onLoad: function (options) {

    this.getData();

    db.collection("demolist").watch({

      onChange:function(res){

        console.log(res.docs)

        //this.setData({

          dataArr : res.docs

        //})

      },

      onError:function(err){

        console.log(err)

      }

    })

  },

  getData(){

    db.collection("demolist").get()

    .then(res=>{

      this.setData({

        dataArr : res.data

      })

    })

  }

在onLoad里面好像不用this.setData

查询的条件

.limit(数字)

可以限制查询多少个，可以用在分页显示等地方。

排序

.orderBy(“要根据哪个排序”,”排序方式”)

.orderBy("author","asc")

（顺序不重要，不过最好写在get前）

asc顺序排序，从小到大

desc倒序，从大到小（通常用这个多，重要的在上，不重要的在下）

.skip(数字)

常常用于分页，数字代表过滤多少个。

比如第一页是1-10号，第二页展示的时候就可以用skip(10)跳过前10个来继续排序

.field(对象)

里面存的是一个对象

如果你想要某几项出现，就用field把该项设为true，就会只返回该项

    db.collection("demolist").field({

      author:true

    }).get()

    .then(res=>{

      console.log(res)

    })

只返回author（id也会返回，唯一标识符）

Command

const db = wx.cloud.database();

const \_ = db.command;

.eq()

db.collection("demolist").where({

      author: \_.eq("张三")

    })

这个操作与author:”张三”

是一样的，还可以有大于小于等。（查看文档）

注意，eq是一个方法。

neq不等于

it小于

ite小于等于

gt大于

gte大于等于

最好是对数字类型进行比较，不要用字符串之类的比较

.in(数组)

表示筛选的数字在该数组范围内

hits: \_.in([255,666])

在255到666范围内显示（255和666都会有）

nin()

不在该数组内

and

hits: \_.and(\_.gt(0),\_.lt(100))

大于0，小于100

or或操作

not

nor

用and来查询多条数据

db.collection('todo').where(\_.and([

{

progress: \_.gt(50)

},

{

tags: 'cloud'

}

])).get()

开头要加一个中括号，代表数组

数组里面有两个对象，分别是progress的值大于50，且tags为cloud

或，非等操作类似

字段操作符

exists

db.collection('todos').where({

tags: \_.exists(true)

})

如果有tags这个字段，则会显示出来，如果没有该字段就不会显示

（注意版本要在2.8.3以上）

mod取余运算

数组查询操作

size

数组长度要满足给出的数字

db.collection('todos').where({

places: \_.size(2)

})

找出 tags 数组字段长度为 2 的所有记录

all

db.collection('todos').where({

tags: \_.all(['cloud', 'database'])

})

如果在tags中包含cloud和database的，就会显示。

顺序不重要，可以反过来

elemMatch

数组里面有对象的才能用

db.collection('todos').where({

places: \_.elemMatch({

area: \_.gt(100),

age: \_.lt(2),

})

})

places数组中有几个对象，这里需要对象的area大于100，age小于2

更新

inc

const \_ = db.command

db.collection('todos').doc('todo-id').update({

data: {

progress: \_.inc(10)

}

})

inc（数字）表示自增数字内的数量，比如这里就是自增10

数字可以为正，也可以为负

mul自乘

remove删除某一属性

db.collection('todos').doc('todo-id').update({

data: {

style: \_.remove()

}

})

删除style属性

set将字段设置为指定的值

会把旧字段覆盖

更新数组

push

db.collection('todos').doc('doc-id').update({

data: {

tags: \_.push(['mini-program', 'cloud'])

}

})

尾部增加数据，会直接在tags后面加入mini-program和cloud

就算没有数组，也是可以创建一个新数组，然后添加

（注意，push里面是一个数组）

push还有很多用法，参考文档

pop

db.collection('todos').doc('doc-id').update({

data: {

tags: \_.pop()

}

})

对最末尾的数据进行删除

unshift 往数组头部增加一个或多个值

db.collection('todos').doc('doc-id').update({

data: {

tags: \_.unshift(['mini-program', 'cloud'])

}

})

shift删除第一个元素

db.collection('todos').doc('doc-id').update({

data: {

tags: \_.shift()

}

})

pull 给定一个值或一个查询条件，将数组中所有匹配给定值或查询条件的元素都移除掉。

db.collection('todos').doc('doc-id').update({

data: {

tags: \_.pull('database')

}

})

就会自动把tags中的database删除

pull也有很多种删除方式，看文档

云函数

创建云函数

右键点击cloudfunctions，先指定当前环境，再新建nodejs的云函数

index.js

在cloud.init()下面，定义数据库

const db = cloud.database();

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

  return await db.collection("demolist").get()

}

await 等待异步请求返回

当把云函数修改完成后，一定要右键上传（云依赖那个）

  onLoad: function (options) {

    wx.cloud.callFunction({

      name:"getData"

    }).then(res=>{

      console.log(res)

    })

  },

在onload中，用wx.cloud.callFunction来获取函数名

数据在res.result.data中

这样就可以不用前端请求数据库，可以在云函数操作

前端数据传到云函数

    wx.cloud.callFunction({

      name:"getData",

      data:{

        name:"张三",

        age:18

      }

    })

在callFunction中，data就是可以传递数值

在云函数中，event就是存传递过去的值

event.name等

云开发p20

C:\Users\asus\WeChatProjects\miniprogram-1