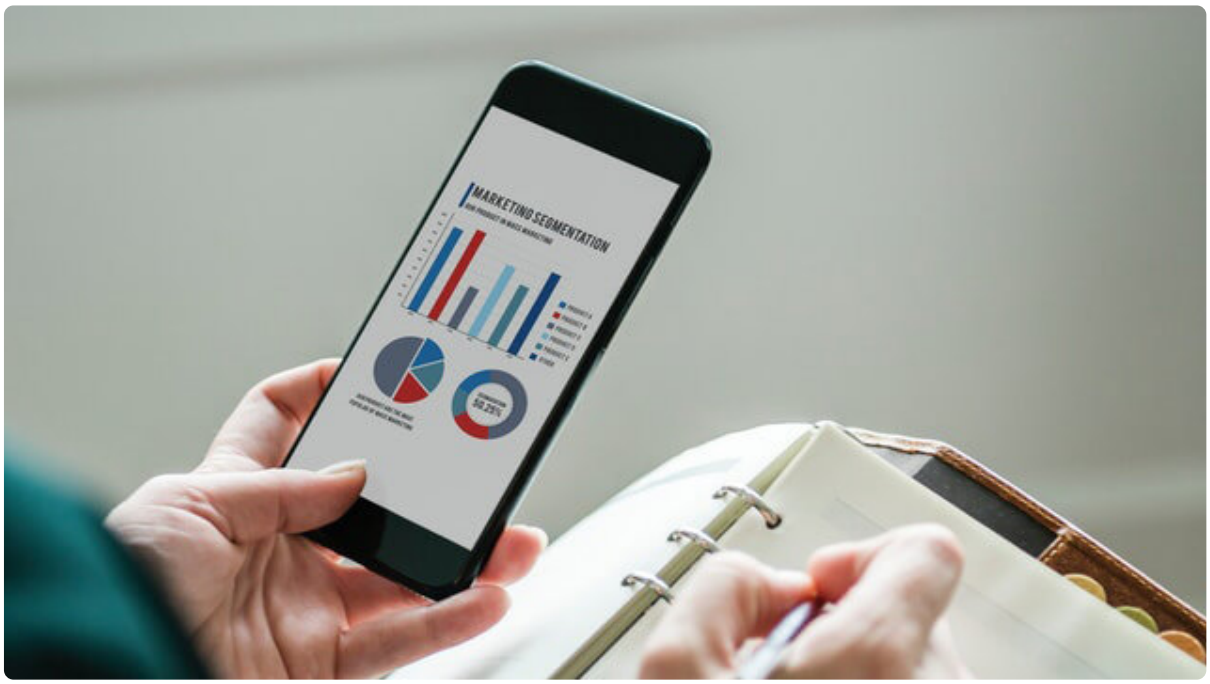
# 08 数据交互：云开发让数据库操作如此简单

更新时间：2019-07-11 15:01:05




> 学习从来无捷径，循序渐进登高峰。
>
> —— 高永祚

上一节我们实践了使用“分类拆解法”的思维，利用小程序生态丰富的组件，开发一个小程序页面。

在实际小程序应用开发中，除了开发手机端的小程序页面，还需要一个服务端来向小程序页面提供显示数据。如果我们把所有显示数据都直接写在小程序页面中，每次我们改变任何一个数据，我们都必须修改小程序页面，然后发布一个新版本的小程序，这显然是不现实的。

上一节的小程序页面中幻灯片具体显示的图片是什么，三个板块菜单点击后显示的具体内容是什么，这些显示数据都需要小程序页面从服务端获取。

因此，本节我们将讲解如何使用云开发实现服务端的数据存储，以及小程序手机端如何实现与服务端的数据交互。

第一个实践内容是页面访问计数功能，如图 10 所示。

**图 10 页面访问计数Demo**


## 1. 页面访问计数功能分析

页面访问计数要实现的功能是，当用户打开一次页面，我们就要记录一次该用户打开页面的次数，并将该用户打开页面的总次数显示在页面底部（图 10 红框部分）。

举例来说，当张三第一次打开页面，页面应该显示“这是你第 1 次访问本页面”；当张三再次打开页面，页面应该显示“这是你第 2 次访问本页面”。而当李四第一次打开页面，页面应该显示“这是你第 1 次访问本页面”。

| 用户ID | 打开页面次数 |
| --- | --- |
| 张三 | 2 |
| 李四 | 1 |

因此，我们可以总结出页面访问计数功能有以下规则：

- 规则 1：每个用户打开页面的次数要单独计算，即每个用户打开页面的次数要使用一条单独的数据记录保存；
- 规则 2：需要一个数据库集合 `counters` 来存储每个用户打开页面次数的数据记录；
- 规则 3：每条数据记录至少要包含两个信息，用户的 ID 和 打开页面的次数；
- 规则 4：当用户第一次打开页面，需要在数据库集合中添加一条该用户打开页面次数的数据记录，并设置打开页面的次数为 1；
- 规则 5：当用户再次打开页面，该用户数据记录的打开页面次数加 1；
- 规则 6：将用户打开页面次数的记录数据显示在页面中。

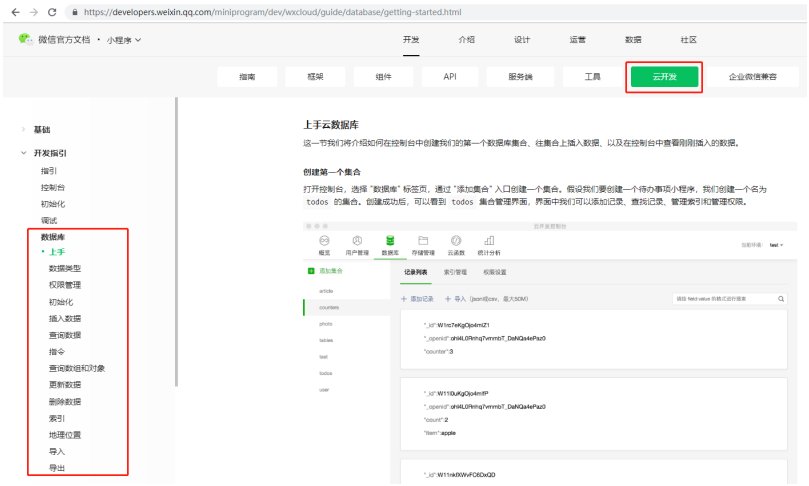
## 2. 建立云开发数据库

在第二章第一节的实践环节，已经请各位同学先行阅读小程序云开发“基础”章节。如你还未完成该实践环节，请暂停阅读本节内容，完成该实践环节，在微信开发者工具中建立好一个已开通云开发的项目，再继续学习本节后续内容。

使用云开发建立数据库非常简单，微信官方的“小程序开发文档”有详细的图文教程，**云开发数据库教程位置如下：**

- **入口网址：云开发数据库教程，如果微信官方文档改版导致链接失效，请按图索骥。**
- **入口位置：在“小程序开发文档”的“云开发”栏目，如图 11 红框标出部分。**

**图 11 云开发数据教程位置**




按照云开发数据库教程“上手”小节介绍的操作步骤，我们首先建立一个数据库集合 `counters` ，完成规则 1 与规则 2 。

## 3. 云开发数据库操作

数据库操作一般分为查询数据、插入数据、更新数据和删除数据。**云开发的数据库操作在图11所示的教程中也有详细的讲解，请仔细阅读、学习。**

规则 3 描述了一条记录要包括用户 ID才能分别存储不同用户的访问次数数据。

通过阅读云开发数据库教程“权限管理”小节的内容，我们知道**云开发数据库中的每一条记录都会自动带上该记录创建者（即小程序用户）的信息。**

所以在我们向数据库集合插入数据时，我们只需要插入打开页面次数 count 这一个信息，云开发数据库会自动加上用户 ID 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**）。

openid 是小程序中用户的唯一标识，每个微信用户都有不同的 openid 。

为了实现规则 6 ，我们需要在 WXML 页面模板中添加一个交互界面：

```
<view>这是你第{{count}}次访问本页面</view>
```

然后，在 JS 逻辑中设置一个数据 count , 用来向界面提供用户访问次数的数据：

```
/**
 * 页面的初始数据
 */
data: {
  count: '' //用于显示的页面访问次数数据
},
```

我们还需要在 JS 逻辑的页面加载事件 onLoad 中定义一个函数 pageCounter ，当用户打开小程序页面时去云开发数据库中获取访问次数：

```
/**
 * 生命周期函数--监听页面加载
 */
onLoad: function(options) {
  //当页面加载时，从数据库集合中获取用户的页面访问次数
  this.pageCounter()
},
```

**小程序页面的系统事件，请参阅微信官方“小程序开发文档”的“框架”->“框架接口”->“页面”->“Page”小节。**

规则 4 与 规则 5 在函数 pageCounter 中具体实现：

```
/**
 * 从数据库集合中获取用户的页面访问次数，并设置用于显示的页面访问次数数据
 */
pageCounter: function() {
  //语句含义见云开发数据库教程 “初始化” 小节
  const db = wx.cloud.database()

  //语句含义见云开发数据库教程 “指令” 小节
  const _ = db.command

  //语句含义见云开发数据库教程 “查询数据” 小节
  db.collection('counters')
    .get()
    .then(res => {

      //判断数据库中是否已有用户的页面访问次数记录，以决定执行规则 4 或 规则 5
      if (res.data.length > 0) {
      //规则 5 ：当用户再次打开页面，该用户数据记录的 打开页面次数 加 1

        //语句含义见云开发数据库教程 “更新数据” 小节
        db.collection('counters').doc(res.data[0]._id).update({
          data: {
            count: _.inc(1)
          }
        })

        //设置用于显示的页面访问次数数据
        this.setData({
          count: res.data[0].count + 1
        })
      }
      else {
      // 规则 4 ：当用户第一次打开页面，需要在数据库表中添加一条该用户打开页面次数的数据记录，
      // 并设置打开页面的次数为 1

        //语句含义见云开发数据库教程 “插入数据” 小节
        db.collection('counters').add({
          data: {
            count: 1
          }
        })

        //设置用于显示的页面访问次数数据
        this.setData({
          count: 1
        })
      }
    });
},
```

在实现代码中详细写明了每一个云开发数据库操作语句的教程位置，请仔细阅读对应内容。

至此，我们就完成了页面访问计数功能的开发。完整源代码获取方式见本节 “5. 专栏源代码”。

## 4. 列表数据显示

接下来我们来看第二个实践内容：页面列表数据显示，如图 12 所示。

图 12 页面列表数据显示 Demo


这个Demo展示了小程序应用中常用的滑动屏幕显示多条列表数据的功能。

#### 4.1 功能分析

列表数据显示需要实现的功能点包括：

- 功能点 1：当页面加载时，从云开发数据库中分页获取第一页数据；
- 功能点 2：当用户下滑屏幕到页面底部时，从云开发数据库中分页获取下一页数据；
- 功能点 3：当云开发数据库中所有数据都已经在页面上显示出来后，用户下滑屏幕将不再访问云开发数据库；
- 功能点 4：将从数据库中获取到的多条数据显示在页面中。

#### 4.2 导入列表数据到数据库

要实现这个功能，我们需要先准备好数据库中的数据。

数据库数据位于专栏源代码的 `demo/pages/05_list_template/list_demo_data.json` 文件，各位同学可以在自己的云开发数据库中新建一个数据库集合 `list_demo` ，然后将数据导入到这个数据库集合。

如何在云开发数据库中导入数据请阅读图 11 所示的云开发数据库教程 “导入” 小节。

**导入数据后请将数据库集合 `list_demo` 的“权限设置”修改为“所有用户可读，仅创建者可读写”。**设置方式请阅读云开发数据库教程 “权限管理”小节的内容。

#### 4.3 实现从数据库分页获取数据

从数据库分页获取数据的实现思路如下：

- 首先定义每次获取多少条数据 `page_count` ；
- 再定义一个变量来记录页面已获取过几次数据 `list_index` ；
- 定义一个数据集合 `title_desc_array` 来保存从数据库中获取到的数据，同时这个数据集合向页面提供显示内容；
- 根据功能点 3 ，还需要定义一个标志 `is_no_more_data` ，来供程序判断是否云开发数据库中的所有数据都已经显示到页面中。

```
/**
 * 页面的初始数据
 */
data: {
  title_desc_array: [], //分页获取到的数据集合，[]表示这是一个数组数据类型
  list_index: 0, //分页获取数据的当前页索引
  page_count: 5, //每次分页获取多少数据
  is_no_more_data: false //记录是否已加载完所有分页数据
},
```

计算 `page_count` 乘以 `list_index` 的值 `offset` ，就可以知道我们已经获取过多少条数据，下次一获取数据应该从 `offset+1` 条开始；

每次获取到数据后，我们需要将 `list_index` 的值增加1，这样下次获取数据才会跳过本次已获取到的数据；

举例来说，当页面加载时， `page_count=5` ， `list_index=0` ，所以 `offset=0` ，从第 1 条数据开始获取，并在获取到数据后设置 `list_index=1` ；

当第二次获取数据时， `page_count=5` ， `list_index=1` ，所以 `offset=5` ，从第 6 条数据开始获取，并在获取到数据后设置 `list_index=2` 。

在每次获取到数据后，我们需要将新获取到的数据放到数据集合 `title_desc_array` 的末尾，这样才能保证页面最底部的数据是最新获取的数据；

如果一次分页查询只返回了 0 行数据，说明云开发数据库中所有数据都已经获取完毕，我们应该设置 `is_no_more_data=true` ， 确保用户后续下滑屏幕到底部时将不再访问云开发数据库，避免对数据库的无谓请求。

有了以上思路，我们就可以写出分页获取数据库数据的函数了，完整函数如下：

```
/**
 * 从云数据库分页获取数据
 */
getTitleDescList() {
  const db = wx.cloud.database()

  //计算已经获取过多少条数据
  var offset = this.data.list_index * this.data.page_count

  var query
  //skip和limit的传入参数必须大于0,
  //因此此处需要对offset等于0（即当页面加载时，从云开发数据库中分页获取第一页数据）的情况进行特殊处理
  if (offset === 0) {
    query = db.collection('list_demo')
      .limit(this.data.page_count) // 限制返回数量为 page_count 条
  }
  else {
    query = db.collection('list_demo')
      .skip(offset) // 跳过已经获取过的数据，从第 offset + 1 条开始返回
      .limit(this.data.page_count) // 限制返回数量为 page_count 条
  }

  //执行云开发数据查询
  query
    .get()
    .then(res => {
      //判断本次查询是否没有获取到数据
      if (res.data.length > 0) {
        //使用数组的 concat 方法将新获取到的数据添加到数据集合末尾
        var title_desc_array = this.data.title_desc_array
        title_desc_array = title_desc_array.concat(res.data)

        this.setData({
          //更新数据集合，使页面显示新获取到的数据
          title_desc_array: title_desc_array,
          //已获取过数据次数加1
          list_index: ++this.data.list_index
        })

      } else {
        //如果查询没有获取到数据，说明云开发数据库中所有数据都已经获取完毕
        //设置数据全部加载完毕的标志
        this.setData({
          is_no_more_data: true
        })
      }
    })
    .catch(err => {
      console.error(err)
    });
},
```

请注意代码中关于 **skip** 和 **limit** 传入值的限制以及处理方式。

#### 4.4 实现页面显示列表数据

在从云开发数据库获取到数据后，我们需要将它们显示在界面上。

这次我们要显示的数据跟以往不同，以往是显示一条数据，这次我们需要显示多条数据。这就需要用 WXML 页面模板的“列表渲染”语法。

WXML 页面模板的语法主要包括单条数据的“数据绑定”，多条数据的“列表渲染”，以及显示隐藏界面的“条件渲染”。在微信官方的“小程序开发文档”中对 WXML 语法有详细解的讲解，请各位同学先阅读、理解官方内容后再继续阅读本节后续内容。

**WXML 语法讲解的文档位置如下：**

- 入口网址：**WXML 语法讲解**，如果微信官方文档改版导致链接失效，请按图索骥。
- 入口位置：在“小程序开发文档”的“框架”栏目，如图 **13** 红框标出部分。

图 **13 WXML** 语法文档位置


在上一节介绍的 **WeUI** 中，有一个用于列表显示的组件 **Panel** ，使用它再结合“列表渲染”可以很简单地实现图 12 所示的列表显示效果：

```
<view class="weui-panel weui-panel_access">
    <view class="weui-panel__bd">
        <block wx:for="{{title_desc_array}}" wx:key="_id" wx:for-item="title_desc">
            <view class="weui-media-box weui-media-box_text">
                <view class="weui-media-box__title weui-media-box__title_in-text">{{title_desc.title}}</view>
                <view class="weui-media-box__desc">{{title_desc.description}}</view>
            </view>
        </block>
    </view>
</view>
```

此外，我们还需要在页面加载时，从云开发数据库中分页获取第一页数据。这需要在页面的 onReady 事件中调用在 4.3 中已编写好的分页获取数据函数 getTitleDescList ：

```
/**
 * 生命周期函数--监听页面初次渲染完成
 */
onReady: function () {
  this.getTitleDescList()
},
```

**4.5 实现用户下滑屏幕到页面底部时刷新数据**

此时，我们还剩一下一个功能点需要实现：当用户下滑屏幕到页面底部时，从云开发数据库中分页获取下一页数据。

当用户下滑屏幕到页面底部，在小程序的页面中会触发一个事件 onReachBottom， 我们需要在这个事件中调用分页获取数据函数 getTitleDescList 。另外，我们还应该判断标志 `is_no_more_data` ，只有当未获取到所有数据的时候，才去数据库中获取新的数据。

```
/**
 * 页面滑动到达底部事件的处理函数
 */
onReachBottom: function () {
  if (!this.data.is_no_more_data) { //如果还有数据未加载完，则获取更多数据
    this.getTitleDescList()
  }
},
```

至此，我们就完成了页面列表数据显示功能的开发。完整源代码获取方式见本节“5. 专栏源代码”。

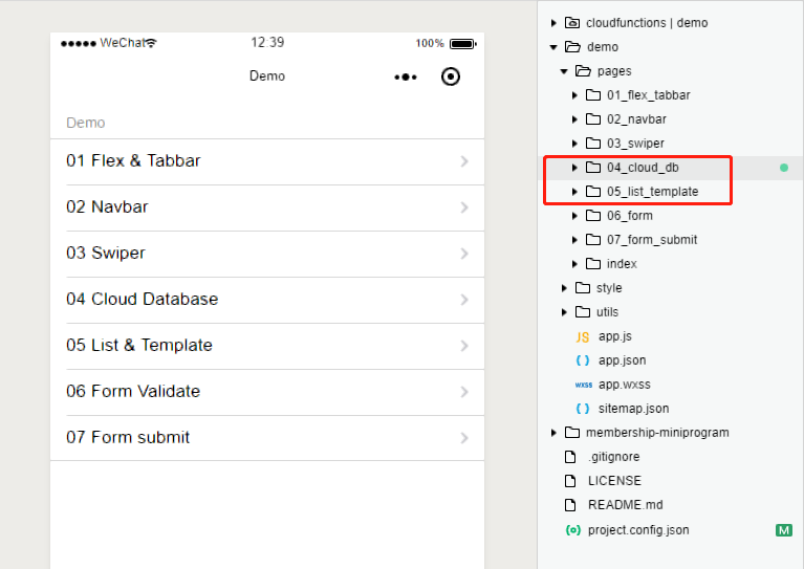
## 5. 专栏源代码

本专栏源代码已上传到 GitHub ，请访问以下地址获取：

https://github.com/liujiec/Membership-ECommerce-Miniprogram

本节源代码内容在图 14 红框标出的位置。

**图 14 本节源代码位置**




## 下节预告

下一节，我们将讲解如何编写输入界面（即表单），如何提交表单，以及云开发中简洁的自动登录能力。

## 实践环节

实践是通往大神之路的唯一捷径。

本节实操内容：

- **使用云开发，编写代码完成图 10 、图 12 所示的页面，如碰到问题，请阅读本专栏源代码学习如何实现。**

07 页面布局：使用组件让页面开发如此简单

09 云函数与表单校验组件让表单提交如此简单