華東師範大學
East China Normal University

基于公共数据库平台的社团应用小程序（赛道一）

课程：云计算系统

姓名：唐小卉

学号：10215501437

2023 年 12 月

目录

1. 设计背景与设计理念

1.1 设计背景

在当前社会背景下，随着科技的不断发展和社会责任感的提升，人们对于动物福利等社会问题的关注逐渐增加。流浪动物的生存状况一直是社会关切的问题之一，同时，大学生群体作为社会中积极向上、关心社会问题的群体，其参与社会公益活动的热情也逐渐高涨。但目前社会上还没有出现良好的、系统的用于群护的应用软件。

目前在校内，社团对喂食的统计依然停留在微信群发照片打卡。因此，研发一款流浪动物喂食管理微信小程序成为开启数字化救助的创新途径。

1.2 设计理念

小程序的设计理念旨在通过数字化和社会化手段，将流浪动物喂食管理转化为一项更具参与性和互动性的社会活动。通过小程序，志愿者能够方便记录自己的喂食工作，而用户则可以实时查看喂食点的情况，形成一个关爱链条。小程序同时整合学校学生信息，使得大学生志愿者更容易参与其中，实现社区和校园的有机连接。同时，为了让小程序服务于更多的想要关注流浪动物的人群，小程序还设有知识手册以及询问途径，方便及时解答。

2. 应用价值与社会意义

2.1 应用价值

该小程序为志愿者提供了方便快捷的平台，使得他们能够记录和分享自己的善举，同时获得善行的认可。应用还提供了实时的喂食点信息和统计数据，激发用户对善举的关注和认可感，促进社区和校园内的正能量传播。此外，小程序的信息模块为用户提供有关流浪动物救助知识，推动社会对于动物福利问题的更深层次关注。

2.2 社会意义

在社会层面上，该小程序通过数字化手段优化了流浪动物喂食管理流程，实现了志愿者、流浪动物和社会的有机连接。志愿者通过小程序更方便地参与到喂食管理中，提高了活动的社会化程度。用户通过了解流浪动物喂食活动，形成对动物福利的正面认知，同时学习相关知识，推动社会形成更加关爱动物的价值观。

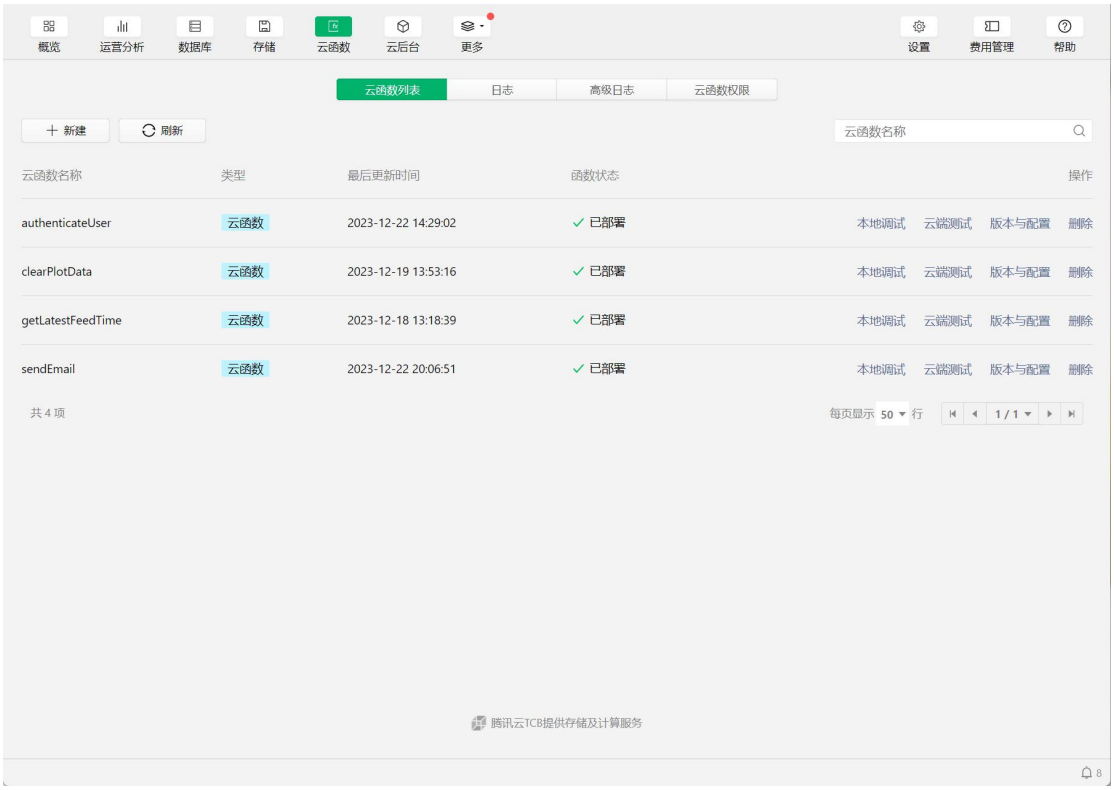
3. 能力集成

本次应用开发最开始想利用公共数据库平台的学生信息接口，了解社团社员或者学校学生的校园卡卡号、密码、姓名等信息，用于信息验证登录小程序。这样可以有效利用学校平台的开发能力，与小程序紧密结合起来，更加方便社团文化组织的身份验证，提供安全可靠的小程序用户体验。

4. 对“云”的使用

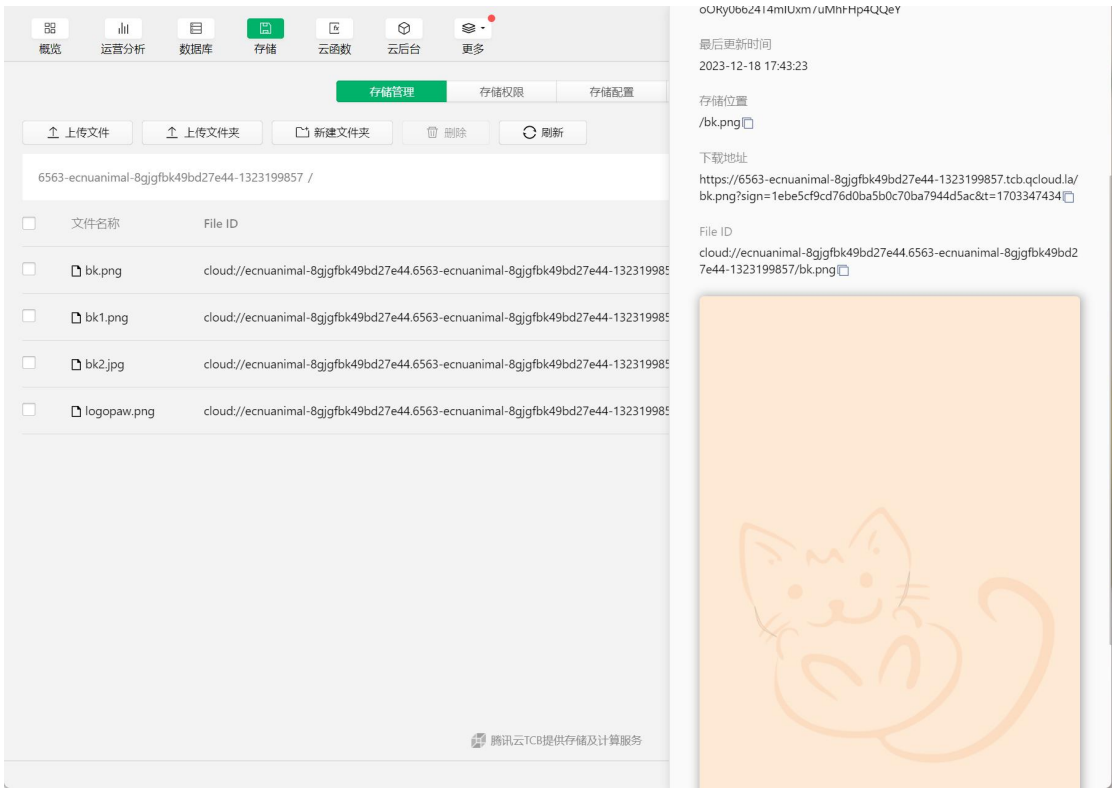
4.1 云函数

本次实验中一共用到了四个云函数，云函数通过本地部署到云端。小程序的云函数可以理解成服务器上面的接口方法，之所以增删改查的逻辑尽量不写在小程序的 js 里面是因为如果要修改小程序需要发版本，而云函数可以随时部署替换，不需要审核发布，更灵活。除此之外还有云函数的专属的方法以及操作数据的权限，小程序 js 是调用不了的。



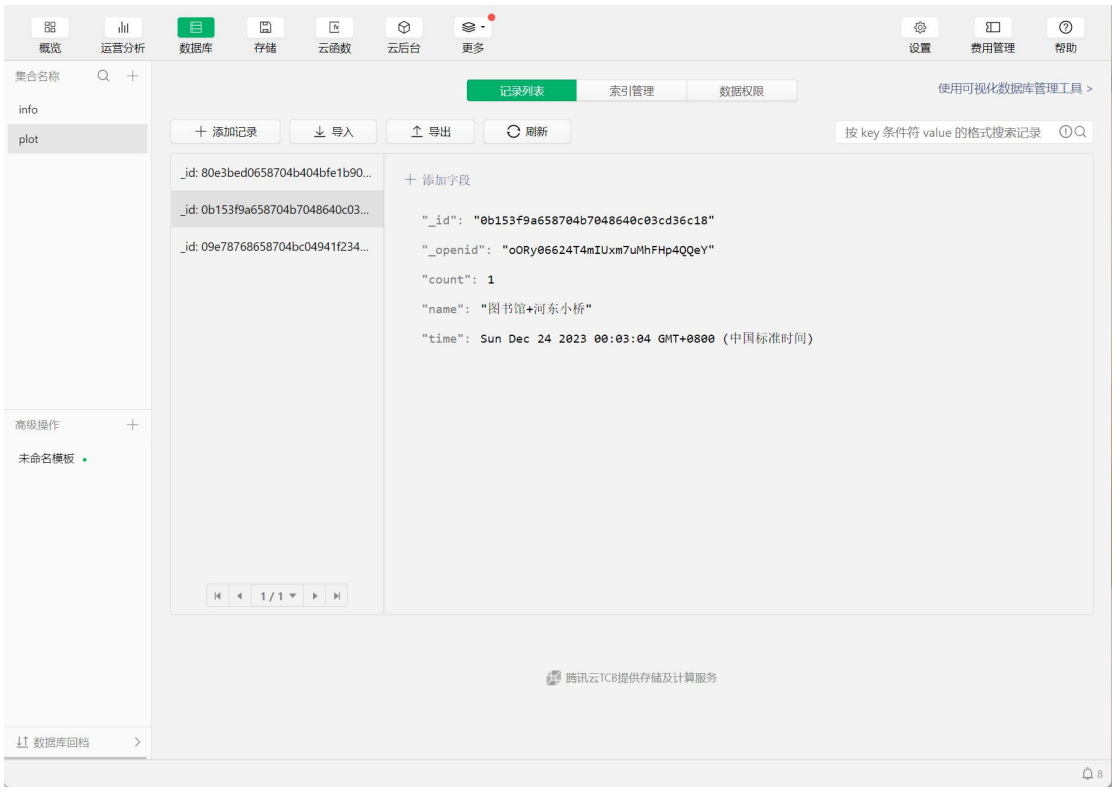
4.2 云存储

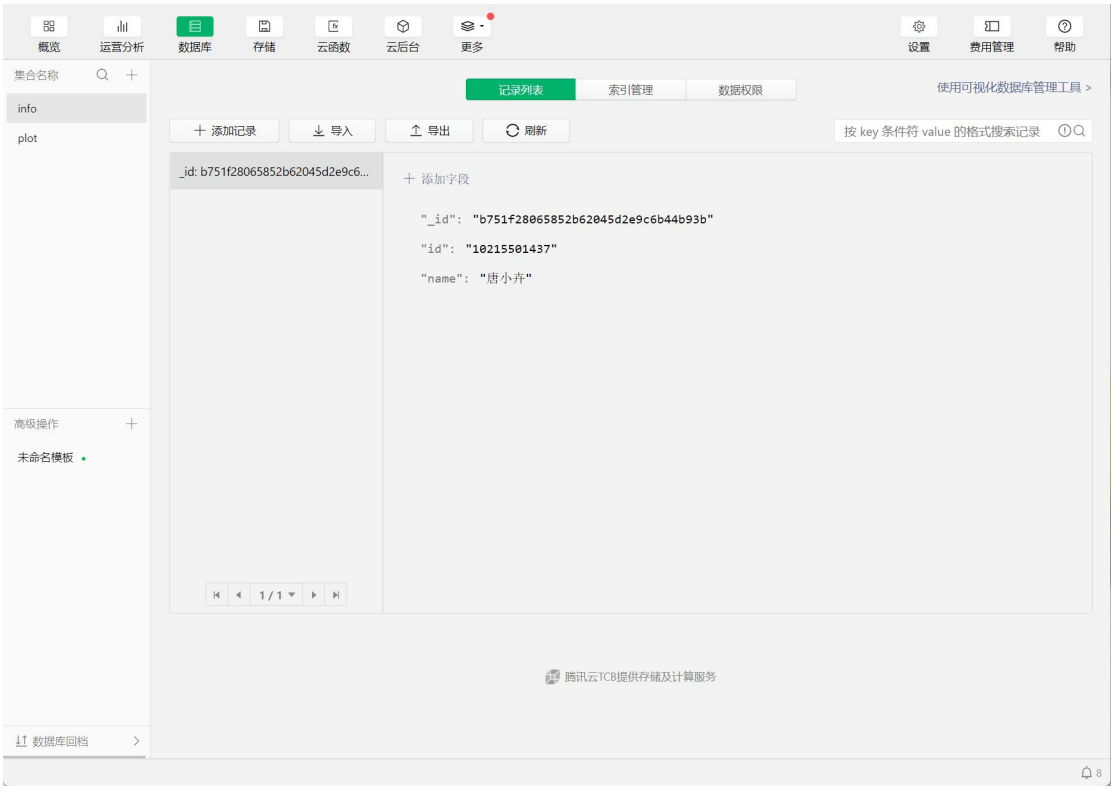
在设计小程序背景图片时，需要访问云端存储照片或者网络上的图片才能插入到界面中，所以我将需要用到的一些背景图上传到云存储空间中。



4.3 云数据库

在云数据库中，有两个集合 plot 和 info，分别用于记录喂食点的喂食情况以及用户信息。





5. 开发过程

首先明确需要开发的功能：
功能 1：根据数据库中的用户信息登录（理论上应该是公共数据库中的学生信息）
功能 2：获取用户的微信头像和昵称
功能 3：用户可以选择自己负责的喂食点进行打卡，打卡记录计入数据库
功能 4：喂食点记录中可以看到今日喂食的次数
功能 5：每天 0 点将喂食记录全部清除
功能 6：用户可以通过小程序发送邮件到动保工作邮箱
功能 7：通过点击跳转到一些知识界面
接下来我将根据功能划分进行开发过程阐述：

5.1 功能 1：根据数据库中的用户信息登录

在云数据库集合 info 中，存储了用户的 name 和 id，预想中应该是连接到公共数据库的接口来获取信息。用户需要通过输入自己的姓名和校园卡号登录到小程序中，才能使用小程序的功能，这也是小程序作为一个社团平台与学校平台的联系。

5.1.1 云函数 authenticateUser

authenticateUser 云函数用于进行用户认证的功能。以下是代码的逻辑和功能解释：

1. 引入了 wx-server-sdk 模块，用于初始化云开发环境。
2. 初始化数据库操作对象 db，并指定要操作的集合为 userInfo，这是一个包含用户信息的集合。
3. 导出了一个名为 main 的异步函数作为云函数的入口函数，接收两个参数 event 和 context。
4. 在入口函数中，首先从 event 参数中获取 name 和 id 两个字段的值，这是用于用户认证的输入数据。
5. 使用 await 关键字，通过在 userInfoCollection 集合中执行 where 查询，查找与输入的 name 和 id 匹配的用户信息。
6. 将查询结果保存在 queryResult 变量中，该结果是一个对象，包含一个 data 字段，其中存储了查询到的用户信息。
7. 判断查询结果的 data 字段的长度是否大于 0，如果大于 0 表示存在匹配的用户信息，即用户认证成功。
8. 如果认证成功，返回一个对象，其中 authenticated 字段为 true。
9. 如果认证失败，返回一个对象，其中 authenticated 字段为 false。
10. 如果在查询过程中出现错误，使用 console.error 记录错误信息，并抛出一个错误。

```
// 云函数入口文件
const cloud = require('wx-server-sdk');


cloud.init();


const db = cloud.database();
const userInfoCollection = db.collection('userInfo');


// 云函数入口函数
exports.main = async (event, context) => {
  const { name, id } = event;


  try {
    // 查询云数据库中是否存在匹配的用户信息
    const queryResult = await userInfoCollection
      .where({
        name: name,
        id: id,
      })
      .get();


    // 如果查询结果不为空，表示用户认证成功
    if (queryResult.data.length > 0) {
      return {
        authenticated: true,
```

```
      };
    } else {
      // 用户认证失败
      return {
        authenticated: false,
      };
    }
  } catch (error) {
    console.error('云函数调用失败', error);
    throw new Error('云函数调用失败');
  }
};
```

5.1.2 其他逻辑

其它逻辑存储在 login 中：

1. 在 Page 函数中定义了一个 data 对象，用于存储页面数据。其中，userInfo 字段用于存储用户信息，authenticated 字段用于表示用户是否认证成功，authFailed 字段用于表示用户认证是否失败，inputName 和 inputId 字段用于存储用户输入的姓名和身份证号码。

2. 定义了两个事件处理函数 bindNameInput 和 bindIdInput，用于当用户在相应的输入框中输入文字时，将输入内容更新到对应的 data 字段中。

3. 定义了事件处理函数 onLogin，用于在用户点击“登录”按钮时触发。该函数调用 wx.getUserProfile 方法获取用户授权，如果用户授权成功，则从返回结果中获取用户信息，并将其保存到全局数据中。

4. 调用 wx.cloud.callFunction 方法调用云函数 authenticateUser，向其传递用户输入的姓名和身份证号码作为参数。如果调用成功，则从返回结果中获取 authenticated 字段的值，判断用户是否认证成功。如果认证成功，则将 authenticated 字段设置为 true，authFailed 字段设置为 false；如果认证失败，则将 authFailed 字段设置为 true。

5. 定义了事件处理函数 navigateToIndex，用于在用户认证成功时跳转到首页。

6. 该代码的功能是实现了一个小程序页面的用户认证和跳转功能。当用户点击“登录”按钮时，会调用 wx.getUserProfile 方法获取用户信息，并将其传递给云函数 authenticateUser 进行认证。如果认证成功，则跳转到首页；如果认证失败，则提示用户认证失败。

```
Page({
  data: {
    userInfo: null,
    authenticated: false,
    authFailed: false,
    inputName: "",
    inputId: "",
  },

  bindNameInput: function (e) {
    this.setData({
```

```
    inputName: e.detail.value,
  });
},

bindIdInput: function (e) {
  this.setData({
    inputId: e.detail.value,
  });
},

onLogin: function () {
  wx.getUserProfile({
    desc: '用于完善会员资料',
    success: (res) => {
      const userInfo = res.userInfo;

      // 将用户信息存储到全局数据中
      getApp().globalData.userInfo = userInfo;

      // 更新页面的数据
      this.setData({
        userInfo: userInfo,
      });

      wx.cloud.callFunction({
        name: 'authenticateUser',
        data: {
          name: this.data.inputName,
          id: this.data.inputId,
        },
        success: (res) => {
          const authenticated = res.result.authenticated;

          if (authenticated) {
            this.setData({
              authenticated: true,
              authFailed: false,
            });
          } else {
            this.setData({
              authFailed: true,
            });
          }
        },
```

```
          fail: (err) => {
            console.error('认证失败', err);
          },
        });
      },
      fail: (err) => {
        console.error('获取用户信息失败', err);
      },
    });
  },

  navigateToIndex: function () {
    wx.switchTab({
      url: '/pages/index/index',
    });
  },
});
```

5.2 功能 2：获取用户的微信头像和昵称

获取用户的微信头像和昵称的功能是微信小程序的内置函数。

```
const userInfo = res.userInfo;

// 将用户信息存储到全局数据中
getApp().globalData.userInfo = userInfo;

// 更新页面的数据
this.setData({
  userInfo: userInfo,
});
```

之后在前端中调用，显示用户的 头像和昵称即可。

```
<view class="avatar" wx:if="{{userInfo}}">
  <image src="{{userInfo.avatarUrl}}" mode="aspectFill"></image>
</view>
<view class="nickName" wx:if="{{userInfo}}">
  {{userInfo.nickName}}
```

5.3 功能 3：用户可以选择自己负责的喂食点进行打卡，打卡记录计入数据库

用户在 feed 界面进行喂食打卡，通过选择自己的喂食点并点击提交按钮实现云数据库的数据更改。后端的逻辑如下：

1. 定义了一个名为 plotOptions 的数组，包含了一些选项，用于在选择框中显示地点选项。
2. 在页面的 data 对象中定义了 selectedPlot 变量，用于保存用户选择的地点，默认选中第一个选项。
3. 实现了 navigateBack 函数，用于点击返回按钮时返回上一页。
4. 实现了 bindPickerChange 函数，当选择框的内容发生变化时，更新 selectedPlot 变量的值。
5. 实现了 submitFeed 函数，用于提交反馈信息。
6. 获取用户选择的地点名称和当前时间。
7. 使用云数据库进行查询，根据地点名称在 plot 集合中查找对应的数据。
   - 如果找到了对应的数据，获取数据的_id，然后更新 count 列数据（加 1）和 time 列（更新为当前时间）。
   - 如果未找到对应数据，则插入新数据，包括地点名称、计数和时间。
8. 根据操作结果显示相应的提示信息。

```
Page({
  data: {
    plotOptions: ["七舍", "牛奶棚", "图书馆+河东小桥", "脑所", "大活+共青场+篮球场", "五舍内", "五舍后", "河西食堂+浴室", "六舍内", "16 舍两侧+电镜中心", "国汉+设计中心"],
    selectedPlot: "七舍", // 默认选中第一个选项
  },

  // 返回按钮点击事件
  navigateBack: function () {
    wx.navigateBack({
      delta: 1, // 返回上一页
    });
  },

  bindPickerChange: function (e) {
    // 选择框内容发生变化时更新 selectedPlot 数据
    const index = e.detail.value;
    const selectedPlot = this.data.plotOptions[index];
    this.setData({
      selectedPlot: selectedPlot,
    });
  },
```

```
submitFeed: function () {
  // 获取选择的地点
  const selectedPlot = this.data.selectedPlot;
  const currentTime = new Date(); // Get current time

  // 在 plot 表中查找对应的数据
  const db = wx.cloud.database();
  db.collection('plot').where({
    name: selectedPlot
  }).get().then(res => {
    // 如果找到了对应的数据
    if (res.data.length > 0) {
      // 获取数据的 _id
      const plotId = res.data[0]._id;
      // 更新 count 列数据和时间列
      db.collection('plot').doc(plotId).update({
        data: {
          count: db.command.inc(1), // count 加 1
          time: currentTime, // 更新时间列
        },
        success: (result) => {
          console.log('更新数据成功', result);
          // 提示提交成功
          wx.showToast({
            title: '提交成功',
            icon: 'success',
            duration: 2000,
          });
        },
        fail: (error) => {
          console.error('更新数据失败', error);
          // 提示提交失败
          wx.showToast({
            title: '提交失败',
            icon: 'none',
            duration: 2000,
          });
        }
      });
    } else {
      console.error('未找到对应的数据');

      // 如果未找到对应数据，则插入新数据
```

```
        db.collection('plot').add({
          data: {
            name: selectedPlot,
            count: 1,
            time: currentTime,
          },
          success: (result) => {
            console.log('插入新数据成功', result);
            // 提示提交成功
            wx.showToast({
              title: '提交成功',
              icon: 'success',
              duration: 2000,
            });
          },
          fail: (error) => {
            console.error('插入新数据失败', error);
            // 提示提交失败
            wx.showToast({
              title: '提交失败',
              icon: 'none',
              duration: 2000,
            });
          }
        });
      }
    }).catch(error => {
      console.error('查询数据失败', error);
      // 提示提交失败
      wx.showToast({
        title: '提交失败',
        icon: 'none',
        duration: 2000,
      });
    });
  },
});
```

5.4 功能 4：喂食点记录中可以看到今日喂食的次数

5.4.1 云函数 getLatestFeedTime

这个功能中用到了云函数 getLatestFeedTime，这个云函数用于查询数据库中每个地点的最近喂食时间信息，并返回结果。代码的逻辑和功能如下所述：

1. 引入 wx-server-sdk 模块，使用`cloud.init`方法初始化云开发环境。
2. 创建数据库实例 db。
3. 导出 main 函数作为云函数的入口函数。
4. 在 main 函数中，使用 await 关键字异步执行以下操作：
   - 调用数据库实例 db 的 collection 方法查询 plot 表，并使用`field`方法指定需要返回的字段（name 和 count）。
   - 将查询结果保存到变量 latestFeedTimes 中。
5. 创建空对象 latestFeedTimesMap，用于存储按名称组织的最近喂食时间数据。
6. 使用 forEach 方法遍历 latestFeedTimes.data 数组，将每个地点的名称和计数存储到 latestFeedTimesMap 对象中。
7. 返回一个包含 code 和 data 的对象，其中 code 为 0 表示查询成功，data 为 latestFeedTimesMap 对象。
8. 如果发生错误，将错误信息打印到控制台，并返回一个包含 code 和 message 的对象，其中 code 为 1 表示查询失败，message 为"查询失败"。

```
const cloud = require('wx-server-sdk');
cloud.init({env: cloud.DYNAMIC_CURRENT_ENV});

const db = cloud.database();

exports.main = async (event, context) => {
  try {
    // 从 plot 表中查询最近的喂食次数
    const latestFeedTimes = await db.collection('plot').field({ name: true, count: true }).get();
    // 将数据按名称组织成对象
    const latestFeedTimesMap = {};
    latestFeedTimes.data.forEach((item) => {
        latestFeedTimesMap[item.name] = item.count;
    });

    return {
      code: 0,
      data: latestFeedTimesMap,
    };
  } catch (err) {
    console.error(err);
    return {
      code: 1,
```

```
        message: '查询失败',
      };
    }
  };
```

5.4.2 其他逻辑

为了更方便管理者查询是否有喂食点还没有进行每日喂食，如果查询不到喂食信息，我们将背景颜色设为红色，如果查询到了信息，就把背景颜色设为绿色。

```
<text class="feed-time" style="{{feedTimes[item] ? 'color: #00CC00; background-color: #EAFBF7' : 'color:
#FF6666; background-color: #FCE4E4'}}">
  {{feedTimes[item] ? '猪咪进食' + feedTimes[item] + '次' : '猪咪还未进食'}}
</text>
```

5.4 功能 5：每天的 0 点喂食记录全部清除

5.4.1 云函数 clearPlotData

云函数 clearPlotData 用于删除数据库集合中的所有文档。代码的逻辑和功能如下所述：

1. 引入 wx-server-sdk 模块，并使用 cloud.init 方法初始化云开发环境。
2. 使用 cloud.database 方法获取数据库引用。
3. 导出一个名为 main 的异步函数作为云函数的入口函数。
4. 在 main 函数中，首先通过 db.collection('plot').where({}).get()方法查询集合中的所有文档，并将结果保存在 result 变量中。
5. 使用 map 方法遍历 result.data 数组，提取每个文档的_id 字段，并将所有_id 保存在 ids 数组中。
6. 使用 Promise.all 方法和 map 方法，将 db.collection('plot').doc(id).remove()方法应用于每个 ids 数组中的 id，以删除对应的文档。
7. 返回一个包含删除成功信息的对象，包括 success 表示是否成功、message 表示操作结果消息、result 表示删除操作的结果。如果出现错误，则返回一个包含失败信息的对象，包括 success 表示是否失败、message 表示错误消息、error 表示错误对象。

```
const cloud = require('wx-server-sdk');
cloud.init({env: cloud.DYNAMIC_CURRENT_ENV});


const db = cloud.database();


// 云函数入口函数
exports.main = async (event, context) => {
  // 获取数据库引用
  const db = cloud.database();


  try {
```

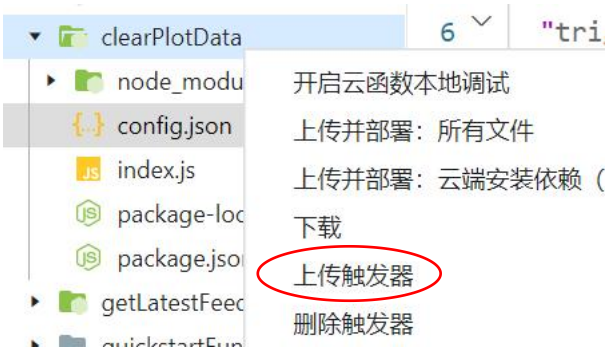
```
    // 删除集合中所有文档
    const result = await db.collection('plot').where({}).get();

    // 获取集合中的所有文档的 _id
    const ids = result.data.map(doc => doc._id);

    // 删除所有文档
    await Promise.all(ids.map(id => db.collection('plot').doc(id).remove()));

    return {
      success: true,
      message: '集合清空成功',
      result: result
    };
  } catch (error) {
    return {
      success: false,
      message: '集合清空失败',
      error: error
    };
  }
};
```

出于某些原因，微信小程序必须使用 id 索引进行搜查，所以删除逻辑看起来比较复杂，

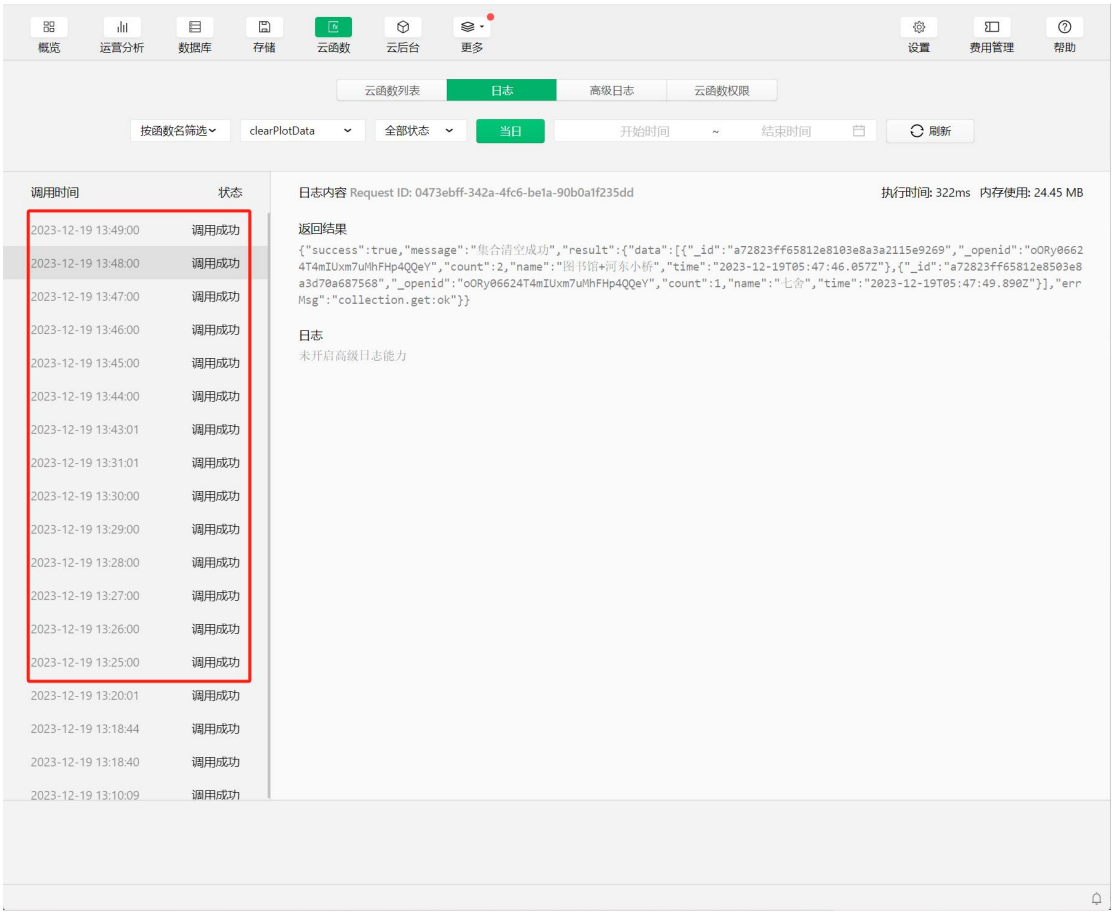
5.4.2 定时触发逻辑

定时出发的逻辑在 config.json 中实现，具体如下：
添加一个 triggers，在每天的 0 时 0 分 0 秒执行云函数，之后向云端传输触发器即可。

```
"triggers": [
  {
    "name": "myTrigger",
    "type": "timer",
    "config": "0 0 0 * * * *"
  }
]
```



因为每天 0 时 0 分 0 秒检测花费的时间周期太长，所以我将每天的 0 秒作为 triggers 进行检测，检测结果如下：



可以看到每个 0 秒都会自动执行 clearPlotData 函数，用于清除数据库中的信息，函数逻辑正确、执行上没有问题。

5.6 功能 6：用户可以通过小程序发送邮件到动保工作邮箱

5.6.1 云函数 sendEmail

在咨询问题的板块，使用 sendEmail 云函数实现发送邮件。云函数使用 SMTP 服务器发送邮件。用户调用该云函数时，需要传递邮件相关的信息，包括发件邮箱、收件人、邮件主题、内容等。云函数会使用 nodemailer 模块创建一个邮件传输对象，并使用配置的 SMTP

服务器信息发送邮件。

1. 引入 wx-server-sdk 模块，并使用 cloud.init 方法初始化云开发环境。
2. 导出一个名为 main 的异步函数作为云函数的入口函数，该函数接收一个 event 参数，用于传递邮件相关的信息。
3. 在 main 函数中，首先引入 nodemailer 模块。
4. 使用 nodemailer.createTransport 方法创建一个邮件传输对象，并配置 SMTP 服务器的相关信息。其中，host 表示 SMTP 服务器地址，port 表示端口号，secure 表示是否使用安全连接，auth 表示授权信息，包括发件邮箱的用户名和授权码（不是 QQ 邮箱的密码）。
5. 创建一个 postMsg 对象，用于存储邮件的相关信息。其中，from 表示发件邮箱，to 表示收件人，subject 表示邮件主题，text 表示纯文本内容，html 表示 HTML 格式的内容。
6. 使用 transporter.sendMail 方法发送邮件，并将结果保存在 res 变量中。
7. 返回 res 作为云函数的执行结果。

```
const cloud = require('wx-server-sdk')
cloud.init({
  env: cloud.DYNAMIC_CURRENT_ENV,
})
exports.main = async (event) => {
  const nodemailer = require("nodemailer");
  let transporter = nodemailer.createTransport({
    host: "smtp.qq.com", //SMTP 服务器地址
    port: 25, //端口号，通常为 465，587，25，不同的邮件客户端端口号可能不一样，网易是 25
    secure: false, //如果端口是 465，就为 true;如果是 587、25，就填 false
    auth: {
      user: "973068733@qq.com",
      pass: "idiyotfbzchsbdgi" //授权码，不是 QQ 邮箱的密码
    }
  });

  let postMsg = {
    from: event.from, //发件邮箱
    to:event.to, //收件人
    subject: event.subject,
    text: event.text,
    html: event.html
  };

  let res = await transporter.sendMail(postMsg);
  return res;
}
```

5.7 功能 7：通过点击跳转到一些知识界面

这个功能比较简单，在组件中设置 navigatetodetail 即可。Navigatetodetail 是微信小程序内置的用于切换页面的函数。以 knowledge（手册）页面为例，小程序会根据不同的切换逻辑切换到对应的界面中。

。

```
navigateToDetail: function (event) {
  const itemName = event.currentTarget.dataset.name;
  const plotNumber = this.mapNameToPlotNumber(itemName);
  wx.navigateTo({
    url: `/pages/knowledge${plotNumber}/knowledge${plotNumber}`,
  });
},

mapNameToPlotNumber: function (name) {
  switch (name) {
    case "社团救助原则（只负责校内）":
      return 1;
    case "遇见小奶猫怎么办":
      return 2;
    case "遇见校内疑似弃猫怎么办":
      return 3;
    case "遇见校内疑似怀孕母猫怎么办":
      return 4;
    case "其他问题":
      return 5;
    default:
      return 0;
  }
},
});
```

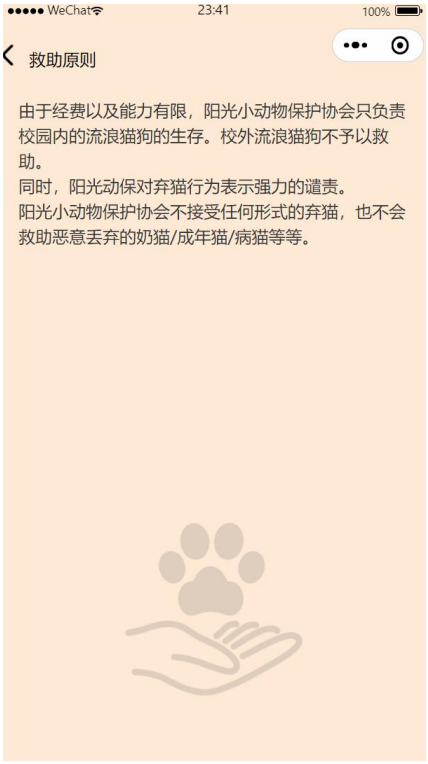
最后展示一下最后的小程序界面：

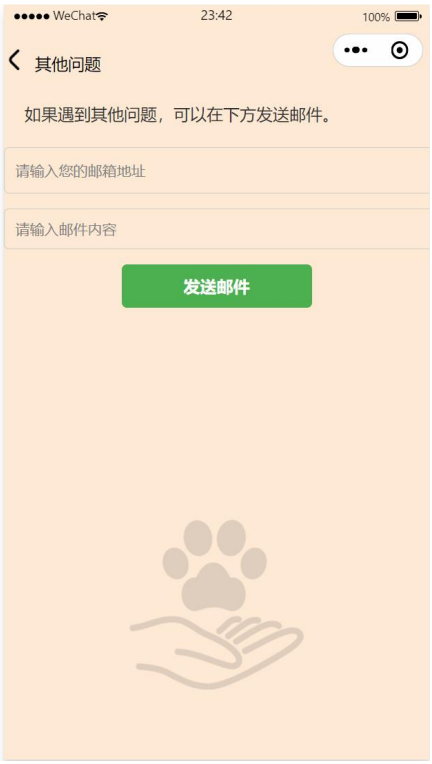

WeChat 23:39 100%
请输入您的姓名
请输入您的校园卡卡号
登录


WeChat 23:40 100%
AAAA迪士尼玩偶批发小唐
喂食打卡
联系我们
喂食点
手册
我的


WeChat 23:41 100%
喂食打卡
选择地点：
七舍
提交
七舍
牛奶棚
图书馆+河东小桥
脑所
取消
确定


WeChat 23:41 100%
文章目录
社团救助原则（只负责校内）
遇见小奶猫怎么办
遇见校内疑似弃猫怎么办
遇见校内疑似怀孕母猫怎么办
其他问题
喂食点
手册
我的


WeChat 23:41 100%
救助原则
由于经费以及能力有限，阳光小动物保护协会只负责校园内的流浪猫狗的生存。校外流浪猫狗不予以救助。
同时，阳光动保对弃猫行为表示强力的谴责。
阳光小动物保护协会不接受任何形式的弃猫，也不会救助恶意丢弃的奶猫/成年猫/病猫等等。


WeChat 23:42 100%
其他问题
如果遇到其他问题，可以在下方发送邮件。
请输入您的邮箱地址
请输入邮件内容
发送邮件


WeChat 23:43 98%
河东 七舍 猪咪进食2次
河东 牛奶棚 猪咪进食1次
河东 图书馆+河东小桥 猪咪进食2次
河东 脑所 猪咪还未进食
河东 大活+共青场+篮球场 猪咪还未进食
河西 五舍内 猪咪还未进食
喂食点
手册
我的


WeChat 23:43 98%
七舍
第七宿舍
第八宿舍
位置在七舍正对面丰巢柜子的背面（表情包所在位置）

6. 技术价值

该小程序充分运用了云开发平台的技术优势，通过云数据库存储志愿者和喂食记录的信息，云存储保存相关图片和多媒体资料，云函数实现了一些后台逻辑的处理。这一技术体系不仅提高了数据的安全性和可靠性，也使得小程序更具扩展性，方便后期功能的迭代更新。与学校接口（未来有机会实现）的结合，使得大学生志愿者信息的获取更加高效，为活动的可持续发展提供了技术支持。

7. 应用成果与反馈

小程序的 demo 制作后，我询问了动保管理层的建议并得到了极高的认可，对于一些不太了解动保的同学需要救助动物的援助时，可以通过小程序最快速的与社团联系。在志愿者群体中进行的小范围调查中，志愿者也认为小程序的使用更加方便快捷，比微信群打卡更能激发动力。

目前鉴于小程序云端资源的价格以及小程序发布上市需要一些其他的条件，暂时不具备上架的条件，或许在未来小程序能够通过用户反馈进行进一步的完善并上架进行使用，最后推广到社会层面，让动物救助科学化、数字化。

总体来说，通过创新的设计理念、强大的应用价值、社会价值和先进的技术价值，该微信小程序为流浪动物喂食管理以及社团管理注入了新的活力。数字化手段的运用不仅使得志愿者活动更加高效和便捷，也促进了社会对于动物福利的认知和关注。技术的引入更是为小程序的可持续发展提供了坚实的基础，为解决社会问题提供了一种创新的路径。

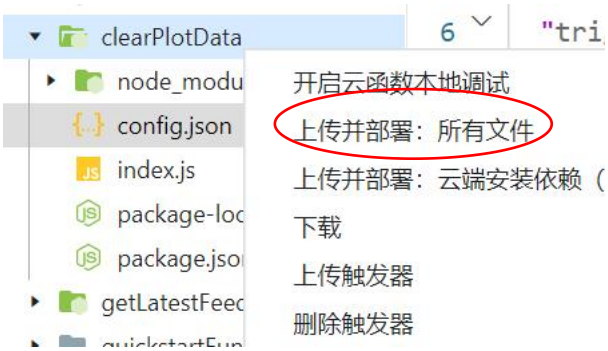
8. 关于项目的补充说明

8.1 一些云技术补充说明

1. 本次项目中，直接使用了微信小程序的云凭条，直接点击云开发即可，就能看到云开发的界面

2. 微信小程序云开发的首月是免费适用，之后需要根据流量进行计费，所以我没有上架小程序，直接在微信中是无法查找到并使用的。

3. 云函数使用必须上传部署到云环境中，只要 index.js 有修改，必须重新部署上传。在对应的文件夹中需要再终端执行：npm install --save wx-server-sdk，在 sendEmail 云函数文件夹中还需要另外执行：npm install nodemailer。



4. 代码以及对应的本地资源已上传至 Github（云资源需要自行部署，并更改某些资源的接口），详情见 https://github.com/TheadoraTang/ECNU-Feed-Animal

8.2 资料参考

1. 微信小程序开发官方文档：https://developers.weixin.qq.com/miniprogram/dev/framework/
2. https://developers.weixin.qq.com/community/develop/article/doc/000e2c97800d70d8fc9a2ce905b813
3. http://t.csdnimg.cn/RzDKo
4. http://t.csdnimg.cn/GxnXc