

浙江工业大学

C++ 课程设计实验报告 (2021级)



实验题目 教职工信息管理系统

学生姓名 葛骏

学 号 202105130406

专业班级 计科 2103

所在学院 计算机科学与技术学院

提交日期 2023 年 5 月 14 日

目 录

1	课程设计题目和内容	1
2	开发测试环境	1
3	实验课题分析	1
4	实验主要模块	2
4.1	链表	2
4.1.1	链表模板类源代码组织	2
4.1.2	链表的实现	2
4.2	教职工类	4
4.2.1	教职工类的定义	4
4.2.2	教职工类的实现	5
4.3	管理类	7
4.3.1	管理类的定义	7
4.3.2	保存和读取的实现	8
4.3.3	数据合法性检测	9
4.4	操作界面	10
4.4.1	GRPC	10
4.4.2	Flutter	10
5	实验调试和运行	11
5.1	实验调试说明	11
5.2	命令行模式	11
5.3	GUI	20
6	实验总结	28

1 课程设计题目和内容

我抽到的题目是第六题，即教职工信息管理系统。要求如下：

基本要求：定义教职工（employee）类，其中至少包括姓名、性别、工号、联系电话、所在学院、系和学历。

1. 设计菜单实现功能选择；
2. 输入功能：输入职工信息，并保存到文件中；
3. 查询功能：
 - 能够根据工号精确查询职工信息；
 - 能够根据姓名、学院、系、学历各项信息查询职工信息；
 - 系进行学历统计，计算各学历的人数；
4. 根据教职工的学历排序输出；
5. 根据工号修改职工信息；
6. 根据工号删除职工信息；
7. 所有的增加、修改、删除能同步到文件；也从文件读取数据到程序。
8. 考虑各项数据的合法性检查

2 开发测试环境

开发测试环境：

- 操作系统：Arch Linux x86_64
- Kernel: 6.3.1-arch1-1
- Shell: fish 3.6.1
- CPU: AMD Ryzen 7 5800H with Radeon Graphics (16) @ 3.200GHz
- GPU: NVIDIA GeForce RTX 3060 Laptop GPU
- Memory: 16GiB
- C++ 编译器: Clang version 15.0.7
- 编辑器: Neovim v0.9.0
- 编码: UTF-8
- 版本控制: Git 2.40.1
- 构建工具: Bazel 6.2
- 其他：
 - C++ 标准: C++ 14
 - libprotoc: 3.21.12
 - flutter: 3.7.12
 - 本文由 typst 编写生成

3 实验课题分析

本题要求实现一个教职工信息管理系统，要求底层的数据结构一定要是链表。那么我将先实现一个双向链表的模板类。

再实现一个教职工类，其中包含需要的信息。

为了实现多种功能，需要专门实现一个管理类。如果用现代的设计模式来说，在 MVC 架构中，这个类就是Controller。

考虑各项数据的合法性检查，需要另外实现一个简单的cpp文件。

4 实验主要模块

4.1 链表

我实现了一个双向链表作为底层数据结构。

4.1.1 链表模板类源代码组织

链表源文件存放在 lib 文件夹中。由于链表是一个模板类，所以所有实现都在头文件中。但是为了定义和实现的分离，使用了 `LinkedList.hpp` 引用 `LinkedListDetail.hpp`。后者再引用 `LinkedListTemplate.hpp`

头文件的引用关系大致如代码 1 所示。

```
// in LinkedList.hpp
#pragma once
#include "LinkedListDetail.hpp"

// in LinkedListDetail.hpp
#include "LinkedListTemplate.hpp"
// detailed implementation ...

// in LinkedListTemplate.hpp
#pragma once
// template implementation ...
```

代码 1 链表头文件引用关系

接下来我将说明链表的实现。

4.1.2 链表的实现

```
#pragma once
// just a simple double-direction linked list
template <typename T> class LinkedList {
public:
    struct Node { // the node of the linked list
        T data;
        Node *next;
        Node *prev;
        Node() : next(nullptr), prev(nullptr){}; // default constructor
```

```

    Node(T data) : data(data), next(nullptr), prev(nullptr){};

};

LinkedList();
LinkedList(const LinkedList &);

LinkedList &operator=(const LinkedList &);

~LinkedList();

void pushBack(T);

void remove(T);

Node *find(T) const;

Node *begin() const;

Node *end() const;

int getSize() const;

private:

Node *head = new Node(); // the head of the linked list.
Node *tail = new Node(); // the tail of the linked list.
int size = 0;           // the size of the linked list.

};

```

代码 2 链表类的定义

如代码 2 所示，我实现的是一个双向链表。每个节点都有一个指向前一个节点和后一个节点的指针。

并且对于每一个链表，含有两个特殊的节点，即头节点和尾节点。这样做的好处是可以方便获取头尾节点，并且在遍历的过程中不会出现空指针。（如果出现空指针则很容易产生 segmentation fault）

链表只给出六个方法：插入方法pushBack，删除方法remove，获取大小方法getSize，查找方法find，头节点begin，尾节点end。

插入方法的实现如代码 3 所示。

```

template <typename T> void LinkedList<T>::pushBack(T data) {
    Node *p = new Node(data);
    p->next = tail;
    p->prev = tail->prev;
    tail->prev->next = p;
    tail->prev = p;
    size++;
}

```

代码 3 插入方法的实现

插入的位置是在尾节点的前面，即tail->prev的位置。

删除方法的实现如代码 4 所示。

```

template <typename T> void LinkedList<T>::remove(T data) {
    Node *p = find(data);
    if (p == nullptr) {
        return;
    }
    p->prev->next = p->next;
    p->next->prev = p->prev;
    delete p;
    size--;
}

```

代码 4 删除方法的实现

删除函数特判了p是否为空指针，如果为空则直接返回。删除的实现是比较简单的，只需要将前后的两节点连接起来即可。不过需要注意的是要手动释放内存，否则会造成内存泄漏。

查找方法的实现如代码 5 所示。

```

template <typename T>
typename LinkedList<T>::Node *LinkedList<T>::find(T data) const {
    Node *p = head->next;
    while (p != tail) {
        if (p->data == data) {
            return p;
        }
        p = p->next;
    }
    return nullptr;
}

```

代码 5 查找方法的实现

查找方法的实现则需要遍历整个链表，直到找到目标节点或者遍历到尾节点。因此查找的时间复杂度为 $O(n)$ ，在后续的实验中，需要注意尽量减少查找的次数。

剩下的两个方法begin和end则是对链表一些节点的直接返回，此处不再赘述。

4.2 教职工类

4.2.1 教职工类的定义

教职工类的定义在src/Employee.hpp中定义了教职工的各种基本信息

```

enum Education { BACHELOR = 0, MASTER = 1, DOCTOR = 2 };

struct EmployeeInfo {
    string name;
    int sex;
    string id;
    string phone;
}

```

```

    string college;
    string department;
    Education education;
    string format() const;
    friend ostream &operator<<(ostream &os, const EmployeeInfo &e);
};

class Employee {
private:
    EmployeeInfo info;

public:
    Employee();
    Employee(const EmployeeInfo);
    Employee(const Employee &e);
    Employee(const string &e);
    EmployeeInfo getInfo() const;
    void setInfo(const EmployeeInfo &e);
    string toString() const;
    bool operator==(const Employee &e) const;
};

```

代码 6 教职工类的定义

教职工类的定义如代码 6 所示。

其中 `EmployeeInfo` 是一个结构体，用于存储教职工的信息。也可以方便参数的传递、返回。

`Employee` 类则是对 `EmployeeInfo` 的封装，提供了一些基本的操作。

`EmployeeInfo` 中的 `format` 方法将教职工格式化为一个字符串，方便进行输出。对 `<<` 的重载则是将这个字符串输出到流中。

而 `Employee` 中的 `toString` 方法则是将教职工的信息转化为便于存储的字符串，用于写入到文件中。常引用 `string` 为参数的构造函数的作用则是将用于存储的字符串转换为 `Employee` 对象。

`Education` 是一个枚举类型，其中

- `BACHELOR = 0` 表示本科
- `MASTER = 1` 表示硕士
- `DOCTOR = 2` 表示博士

4.2.2 教职工类的实现

本节主要介绍教职工类中，用于转储的函数和用于输出的函数的实现。

```

string EmployeeInfo::format() const {
    string res;

```

```

res += "Name: " + name + "\n";
res += "ID: " + id + "\n";
res += "Phone: " + phone + "\n";
res += "College: " + college + "\n";
res += "Department: " + department + "\n";
switch (education) {
    case BACHELOR:
        res += "Education: BACHELOR\n";
        break;
    case MASTER:
        res += "Education: MASTER\n";
        break;
    case DOCTOR:
        res += "Education: DOCTOR\n";
        break;
}
return res;
}

ostream &operator<<(ostream &os, const EmployeeInfo &e) {
    os << e.format();
    return os;
}

```

代码 7 format函数的实现

代码 7 是EmployeeInfo中format函数的实现。该函数将教职工的信息格式化为一个字符串，方便进行输出。

而<<的重载则是将这个字符串输出到流中。

```

Employee::Employee(const string &s) {
    std::stringstream ss(s);
    ss >> info.name;
    ss >> info.id;
    ss >> info.phone;
    ss >> info.college;
    ss >> info.department;
    string education;
    ss >> education;
    if (education == "BACHELOR")
        info.education = BACHELOR;
    else if (education == "MASTER")
        info.education = MASTER;
    else

```

```
    info.education = DOCTOR;  
}
```

代码 8 常引用string为参数的构造函数的实现

代码 8 是常引用string为参数的构造函数的实现。调用该构造函数时，需要传入一个用于存储的字符串，该函数将字符串转换为 Employee对象。

```
std::string Employee::toString() const {  
    string res;  
    res += info.name + " ";  
    res += info.id + " ";  
    res += info.phone + " ";  
    res += info.college + " ";  
    res += info.department + " ";  
    switch (info.education) {  
        case BACHELOR:  
            res += "BACHELOR";  
            break;  
        case MASTER:  
            res += "MASTER";  
            break;  
        case DOCTOR:  
            res += "DOCTOR";  
            break;  
    }  
    return res;  
}
```

代码 9 toString函数的实现

代码 9 是Employee中toString函数的实现。该函数将教职工的信息转化为便于存储的字符串，用于写入到文件中。

4.3 管理类

Management类是相当于MVC加工中Controller的角色。定于与src/Management.hpp中。主要功能是编写业务逻辑和接口。

4.3.1 管理类的定义

```
class Management {  
private:  
    LinkedList<Employee> employees;  
  
public:  
    bool isSaved = false;  
    Management();
```

```

~Management();

int save();
int load();
int addEmployee(const Employee &);

int deleteEmployee(const string &);

int modifyEmployee(const Employee &);

Employee searchEmployeeByID(const string &) const;

LinkedList<Employee> searchEmployee(const string &) const;

string statisticsByDepartment() const;

std::map<std::string, std::map<Education, int>>

statisticsByDepartmentMap() const;

string statisticsByEducation() const;

std::map<Education, int> statisticsByEducationMap() const;

};


```

代码 10 管理类的定义

Management类的定义如代码 10 所示。

- `save` 方法用于将教职工信息保存到文件中。
- `load` 方法用于从文件中读取教职工信息。
- `addEmployee` 方法用于添加教职工。
- `deleteEmployee` 方法用于删除教职工。
- `modifyEmployee` 方法用于修改教职工信息。
- `searchEmployeeByID` 方法用于根据教职工的 ID 查找教职工。
- `searchEmployee` 方法用于根据关键字查找教职工。
- `statisticsByDepartment` 方法用于统计各个学院的教职工学历。
- `statisticsByDepartmentMap` 方法用于统计各个学院的教职工学历，返回一个 `map` 对象。
- `statisticsByEducation` 方法用于统计各个学历的教职工人数。
- `statisticsByEducationMap` 方法用于统计各个学历的教职工人数，返回一个 `map` 对象。

4.3.2 保存和读取的实现

```

const string path = "/home/finley/data.txt"; // TODO: change the path

int Management::save() {

    std::ofstream fout(path);

    if (!fout.is_open()) {

        return false;
    }

    auto *p = employees.begin();

    while (p != employees.end()) {

        fout << p->data.toString() << '\n';

        p = p->next;
    }
}


```

```

    }

    fout.close();
    isSaved = true;
    return true;
}

int Management::load() {
    std::ifstream fin(path);
    if (!fin.is_open()) {
        return false;
    }
    while (!fin.eof()) {
        std::string line;
        std::getline(fin, line);
        if (line.empty()) {
            continue;
        }
        Employee e(line);
        employees.pushBack(e);
    }
    fin.close();
    return true;
}

```

代码 11 保存和读取的实现

代码 11 是Management类中save和load方法的实现。save方法将教职工信息保存到文件中，load方法从文件中读取教职工信息。

需要注意，path变量是保存文件的路径，需要根据实际情况修改。

4.3.3 数据合法性检测

```

#include "validate.hpp"

bool isNameValid(const string &s) { return s.length() > 0; }

bool isIDValid(const string &s) {
    if (s.length() != 8) {
        return false;
    }
    for (int i = 0; i < 8; i++) {
        if (s[i] < '0' || s[i] > '9') {
            return false;
        }
    }
}

```

```

    return true;
}

bool isPhoneValid(const string &s) {
    if (s.length() != 11) {
        return false;
    }
    for (int i = 0; i < 11; i++) {
        if (s[i] < '0' || s[i] > '9') {
            return false;
        }
    }
    return true;
}

bool isDepartmentValid(const string &s) { return s.length() > 0; }

bool isCollegeValid(const string &s) { return s.length() > 0; }

bool validate(const EmployeeInfo &info) {
    return isNameValid(info.name) && isIDValid(info.id) &&
           isPhoneValid(info.phone) && isDepartmentValid(info.department) &&
           isCollegeValid(info.college);
}

```

代码 12 数据合法性检测

代码 12 是数据合法性检测的实现。可以对教职工的姓名、ID、电话、学院、学历进行合法性检测。

4.4 操作界面

命令行界面在UserInterface.cpp中实现。实现代码详见附录。为了实现GUI，我引入了grpc

4.4.1 GRPC

GRPC, (Google Remote Procession Call), 是谷歌开发的一套开源的远程过程调用框架。它使用 Protocol Buffers 作为接口描述语言 (IDL)，支持多种语言，包括 C++、Java、Python、Go、Ruby、C# 等。

使用 GRPC，可以方便地生成后端服务器和前端客户端交互的代码。

后端服务器的代码定义于server.cpp，默认监听50051端口。

4.4.2 Flutter

Flutter 是谷歌开发的一套跨平台的移动应用开发框架。它使用 Dart 语言作为开发语言，支持 Android、iOS、Web、Windows、MacOS、Linux 等多个平台。

本项目的前端代码定义于employee目录下。

5 实验调试和运行

5.1 实验调试说明

需要安装 `bazel` 构建工具。

笔者使用的是 Arch Linux，可以直接使用 `yay -S bazel` 安装。

GUI界面使用flutter实现，需要安装配置flutter。

5.2 命令行模式

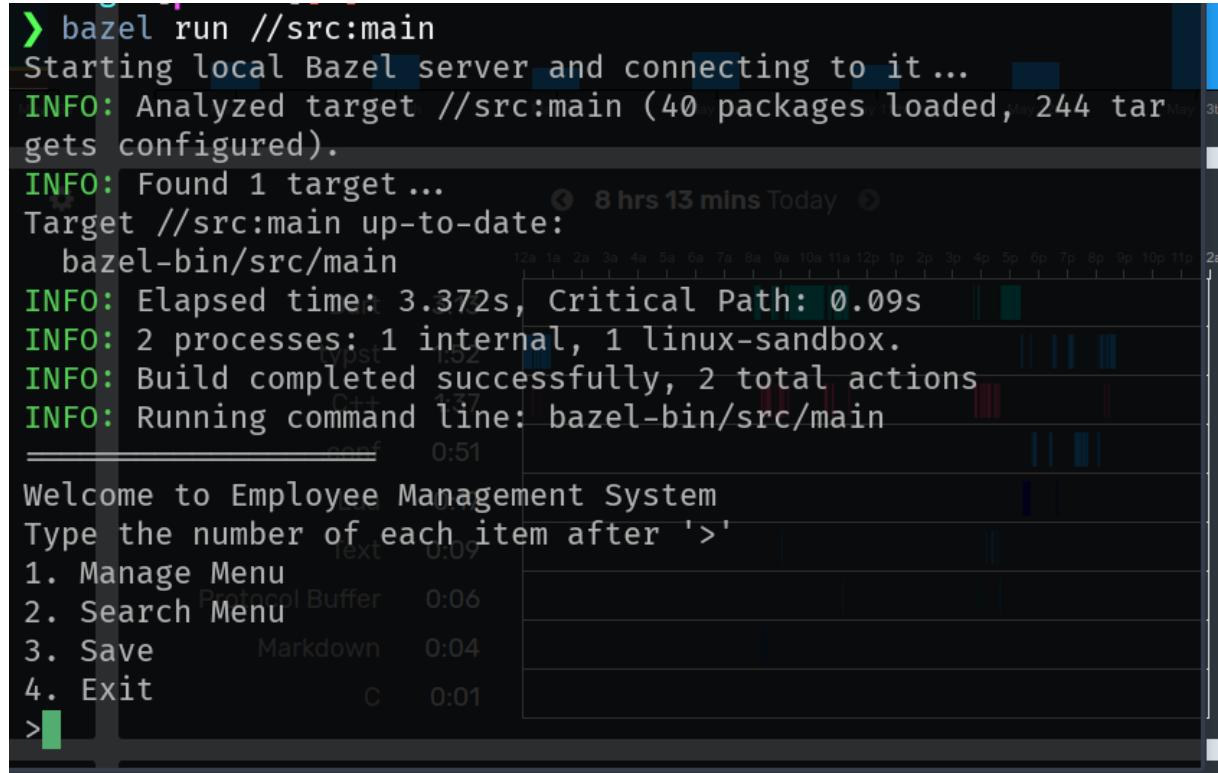
执行编译命令

```
bazel build //src:main
```

等待编译完成后，执行

```
bazel run //src:main
```

进入命令行模式，如果正常进行，则会显示图 1 所示的界面。



```
> bazel run //src:main
Starting local Bazel server and connecting to it ...
INFO: Analyzed target //src:main (40 packages loaded, 244 targets configured).
INFO: Found 1 target ...
Target //src:main up-to-date:
  bazel-bin/src/main
INFO: Elapsed time: 3.372s, Critical Path: 0.09s
INFO: 2 processes: 1 internal, 1 linux-sandbox.
INFO: Build completed successfully, 2 total actions
INFO: Running command line: bazel-bin/src/main

Welcome to Employee Management System
Type the number of each item after '>'

1. Manage Menu
2. Search Menu
3. Save
4. Exit
>
```

图 1 进入命令行模式

键入1则进入管理菜单，如图图 2 所示。

```

INFO: Analyzed target //src:main (0 packages loaded).
INFO: Found 1 target...
Target //src:main up-to-date:
INFO: /src/main (0 ms)
INFO: Elapsed time: 3.372s, Critical Path: 0.0s
INFO: 2 processes: 1 internal, 1 external
INFO: Build completed successfully
INFO: Running command line: bazel run //src:main

```



```

Welcome to Employee Management System
Type the number of each item after '>'
1. Manage Menu
2. Search Menu
3. Save
4. Exit
>1
12
Manage Menu
Type the number of each item after '>'
1. Add
2. Delete
3. Modify
4. Back
>

```

图1 进入

5.3 GUI

编译后端服务器

bazel build //src:server

运行后端服务器

bazel run //src:server

进入前端目录employee 编译运行前端

图2 管理菜单

再键入1则可以添加教职工

```

Welcome to Employee Management System
Type the number of each item after '>'

1. Add
2. Delete
3. Modify
4. Back

>1

ID: 12345678
Name: Finley
Phone: 18899993333
Department: CS
College: CS
Education: 0 for bachelor, 1 for master, 2 for doctor

2

Success

```



```

Welcome to Employee Management System
Type the number of each item after '>'

1. Manage Menu
2. Search Menu
3. Save
4. Exit

>

```

图1 进入命令行模式

键入1则进入管理菜单，如图图2所示。

图3 添加教职工

键入3则保存当前教职工信息到文件中。

```

1: bazel run //src:main ~\L/c/De X 2: vim data.txt ~ X +

```

	data.txt	X
1	Finley	12345678 18899993333 CS CS DOCTOR

图4 保存教职工信息

图4为保存输出的文件。

接下来测试读取文件。先键入4退出程序

重新运行后，测试查询功能。键入2进入查询菜单。

再键入1通过ID查询教职工信息。

The screenshot shows two terminal windows of the Employee Management System.

Left Terminal Window:

- Shows the main menu with options: 2. Search Menu, 3. Save, 4. Exit.
- Submenu for 'Search Menu' is displayed, with option 2 highlighted.
- Input prompt: >2
- Output: C++ 程序设计实验报告 (2021版)
- Search Menu sub-menu:
 - Type the number of each item after '>'
 - 1. Search by ID
 - 2. Search by name, phone, department or college.
 - 3. Statistics educational background of each department
 - 4. Statistics number of employees each educational background
 - 5. Back
- Input: >1
- Output: ID: 12345678
- Employee found
- Name: Finley
- ID: 12345678
- Phone: 18899993333
- College: CS
- Department: CS
- Education: DOCTOR

Right Terminal Window:

- Shows the main menu with options: 1. Manage Menu, 2. Search Menu, 3. Save, 4. Exit.
- Input prompt: >3
- Output: 钮入3 则保存当前教职工信息到文件中。
- Success message: Success
- Employee information output (same as left window):
- Welcome to Employee Management System
- Type the number of each item after '>'
- 1. Manage Menu
- 2. Search Menu
- 3. Save
- 4. Exit

图5 查询教职工信息

接下来测试修改功能。键入1进入管理菜单。然后3进入修改菜单。

```

Manage Menu
Type the number of each item after '>'

1. Add
2. Delete
3. Modify
4. Back

>3
ID: 12345678
Name: Finley
New name:fff
Phone: 18899993333
New phone:11122223333
Department: CS
New department:ee
College: CS
New college:ee
Education: 2
New education:Education: 0 for bachelor, 1 for master, 2 for doctor

1
Success

```



```

Welcome to Employee Management System
Type the number of each item after '>'

1. Manage Menu
2. Search Menu
3. Save
4. Exit

>3

1: bazel run //src:main ~L/c/De X 2: vim data.txt ~ + 

1. data.txt X
1 Finley 12345678 18899993333 CS CS DOCTOR

```

图3 添加教职工

键入3则保存当前教职工信息到文件中。

图4 保存教职工信息

接下来测试读取文件。先键入4退出程序

图6 修改教职工信息

修改后，保存，查看文件。

```

$W [2/2] vim data.txt ~ $W

1: bazel run //src:main ~L/c/De X 2: vim data.txt ~ + 

1. data.txt X
1 fff 12345678 11122223333 ee ee MASTER

```

图7 修改后文件信息

测试删除功能。键入1进入管理菜单。然后2进入删除菜单。输入ID，删除教职工信息。

```

4. Exit
>1
Manage Menu
Type the number of each item after '>'
1. Add
2. Delete
3. Modify
4. Back
>2
ID: 12345678
Success
=====

Welcome to Employee Management System
Type the number of each item after '>'
1. Manage Menu
2. Search Menu
3. Save
4. Exit
>

```

运行后端服务器
bazel run //src:server
进入前端目录tempolyee 编译运行前端
flutter run

6 实验总结
15

图 8 删除教职工信息

再次查看文件，可以看到教职工信息已经被删除。

```

$W [2/2] vim data.txt ~
1: bazel run //src:main ~L/c/De X 2: vim data.txt ~ X +
1. data.txt X
1
~ 湖江工大
~ C++ 课程设计实验报告
~ (2021版)
~ 
~ Manage Menu
~ Type the number of each item
~ 1. Add
~ 2. Delete
~ 3. Modify

```

图 9 删除后文件信息

为了测试统计功能，先添加一些随机数据。我写了一个python脚本用于生成随机数。

```
#!/bin/env python3
import random
```

```
used_id = set()

def generate_test_data() -> str:
    # random data
    name:str = "name" + str(random.randint(10000000, 100000000))
    id:str = str(random.randint(10000000, 100000000))
    while id in used_id:
        id = str(random.randint(10000000, 100000000))
    used_id.add(id)

    phone:str = "1" + str(random.randint(1000000000, 10000000000))
    department:str = random.choice(("CS", "EE", "Math"))
    college:str = random.choice(("CS", "EE", "Math"))
    edu:str = random.choice(("MASTER", "BACHELOR", "DOCTOR"))

    data = "{0} {1} {2} {3} {4} {5}".format(name, id, phone, department, college, edu)
    return data

def main():
    for i in range(20):
        print(generate_test_data())

if __name__ == "__main__":
    main()
```

代码 13 生成随机数据脚本

	name	id	score	major	degree
1	name64081034	56265375	15425924388	EE	Math BACHELOR
2	name61146387	49952827	11575132105	CS	Math MASTER
3	name93987569	82089673	18013837642	EE	EE BACHELOR
4	name23789943	40460182	13730418683	CS	CS DOCTOR
5	name93221166	83444934	12274595918	EE	CS DOCTOR
6	name77304469	77433894	19890563460	Math	Math MASTER
7	name51535897	82579866	17579921645	EE	Math BACHELOR
8	name49816438	35456584	17217262127	CS	EE BACHELOR
9	name23805008	17648401	16606532088	EE	Math BACHELOR
10	name70727579	68458552	15806561685	Math	CS MASTER
11	name92064739	65830527	13325076432	Math	Math MASTER
12	name43191402	11158379	11000960989	EE	CS BACHELOR
13	name50183951	82907936	16741791819	CS	CS DOCTOR
14	name10627334	24987891	16411526226	Math	CS MASTER
15	name61784833	72199159	11295850620	CS	EE MASTER
16	name83387747	38065653	16715172266	EE	CS DOCTOR
17	name93543568	23192550	11747497340	CS	CS DOCTOR
18	name46110377	89713572	14242861287	EE	CS MASTER
19	name33901312	29073890	16542856082	EE	CS DOCTOR
20	name25040060	46534147	12804018670	EE	Math BACHELOR

图 10 随机数据

测试统计功能。

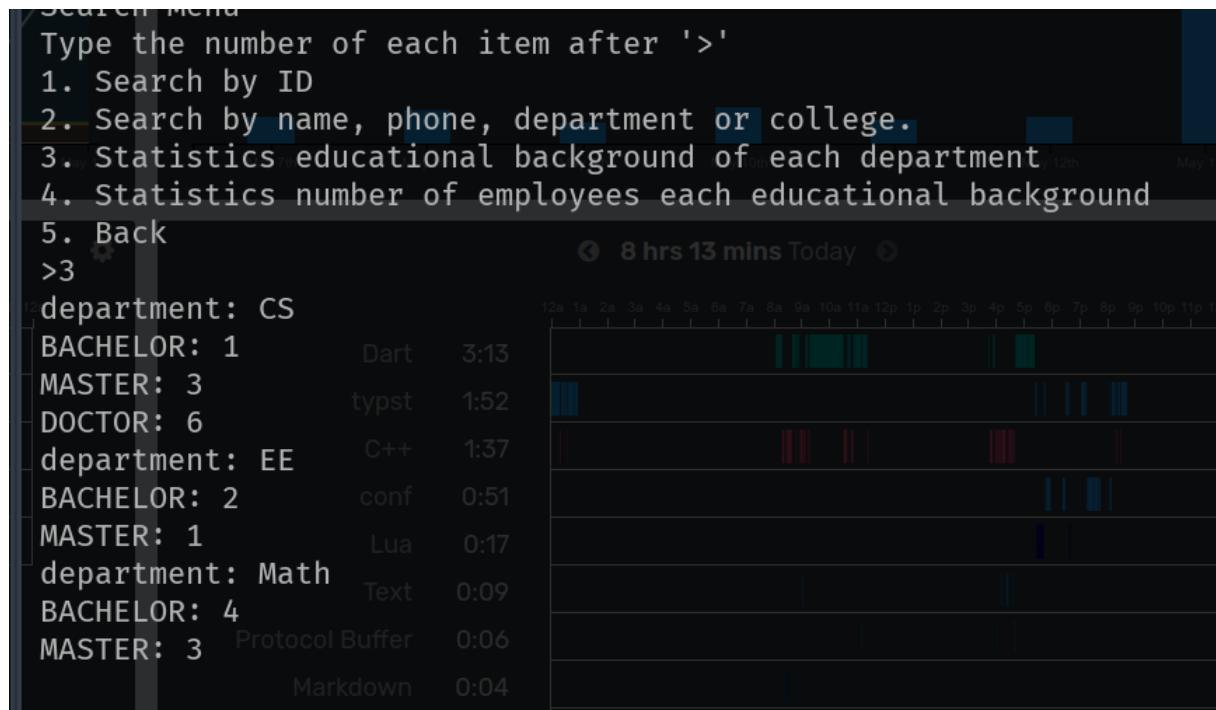


图 11 统计功能: 各专业不同学历人数

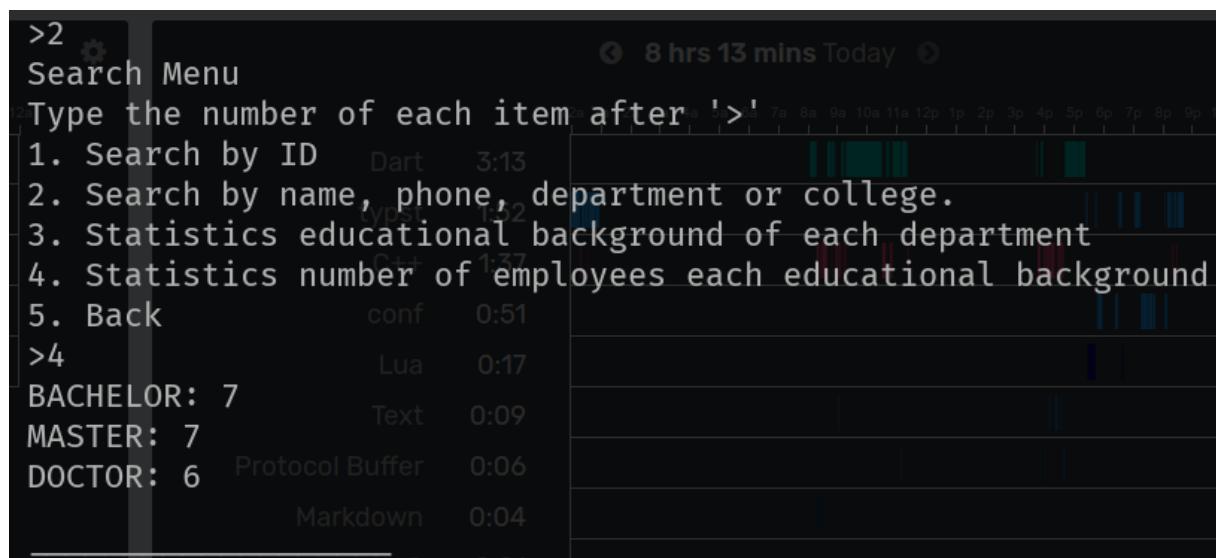


图 12 统计功能: 全体不同学历人数

测试数据合法性检查。

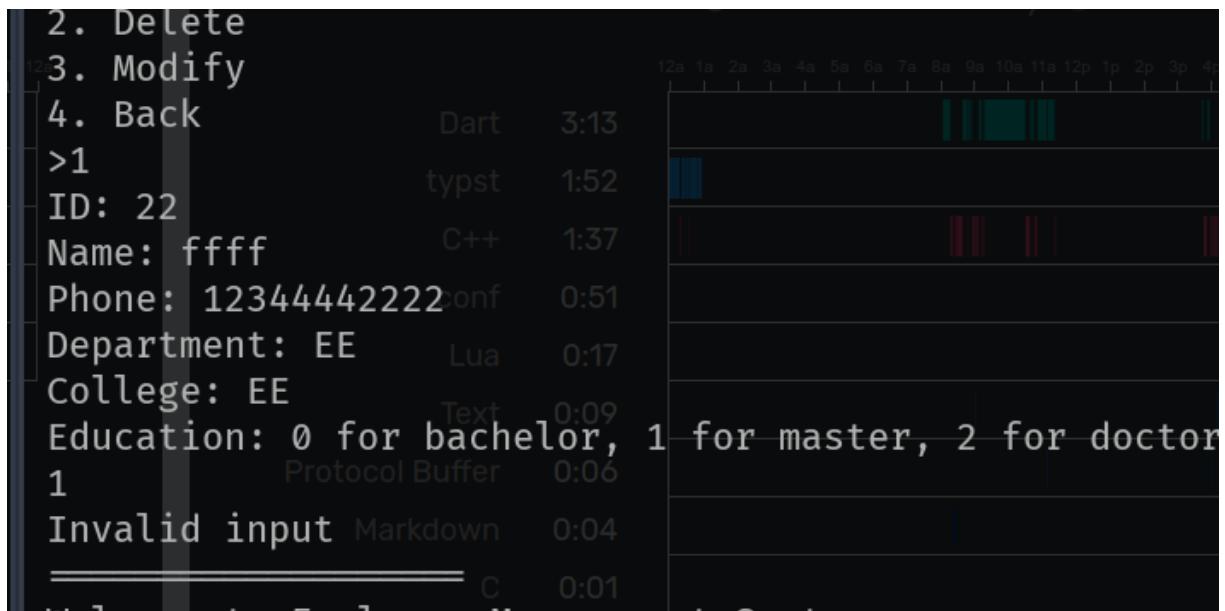


图 13 数据合法性检查

5.3 GUI

编译后端服务器

```
bazel build //src:server
```

运行后端服务器

```
bazel run //src:server
```

进入前端目录employee 编译运行前端。

```
flutter run
```

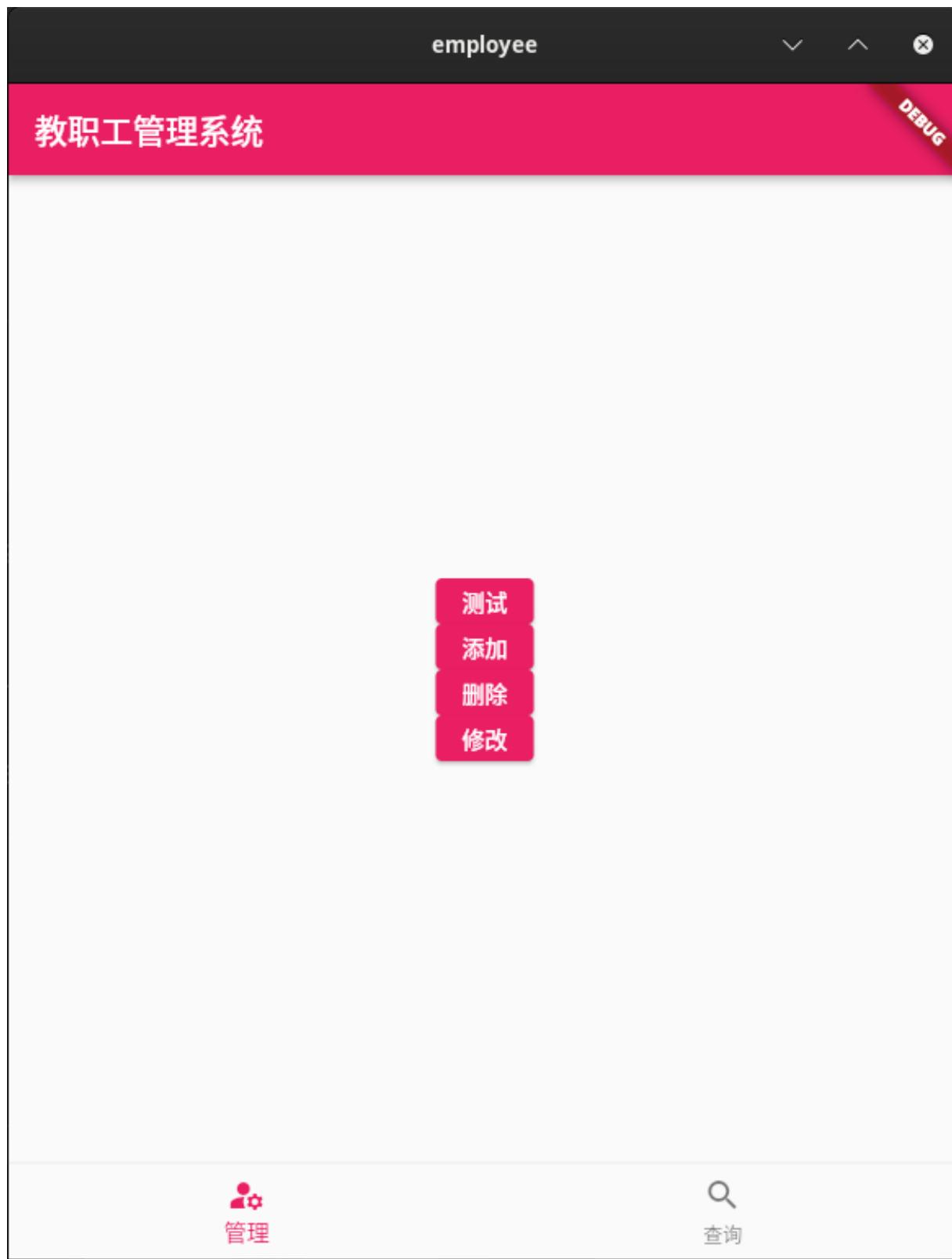


图 14 启动前端

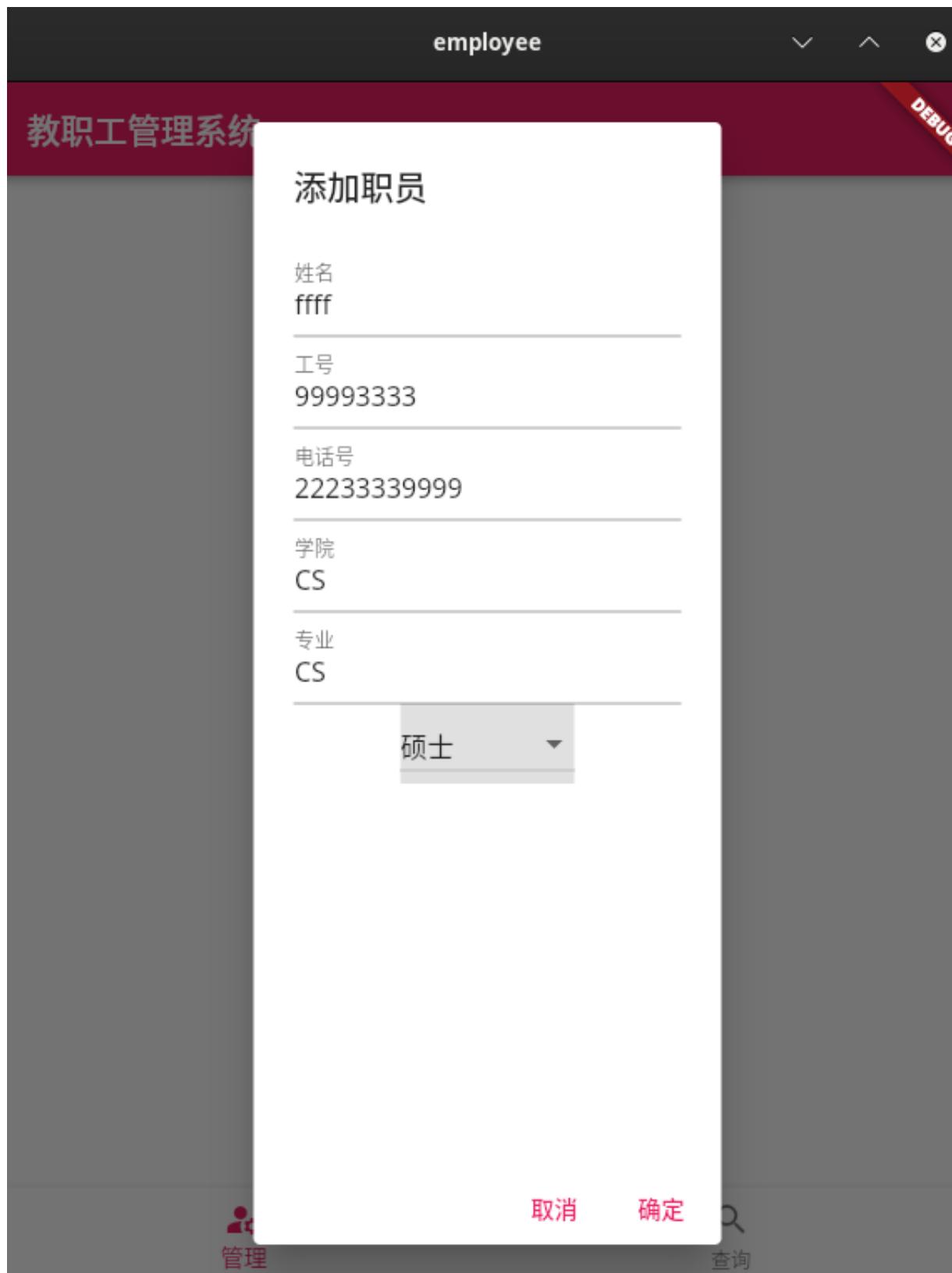


图 15 添加教职工



图 16 删除教职工



图 17 修改教职工



图 18 查询教职工

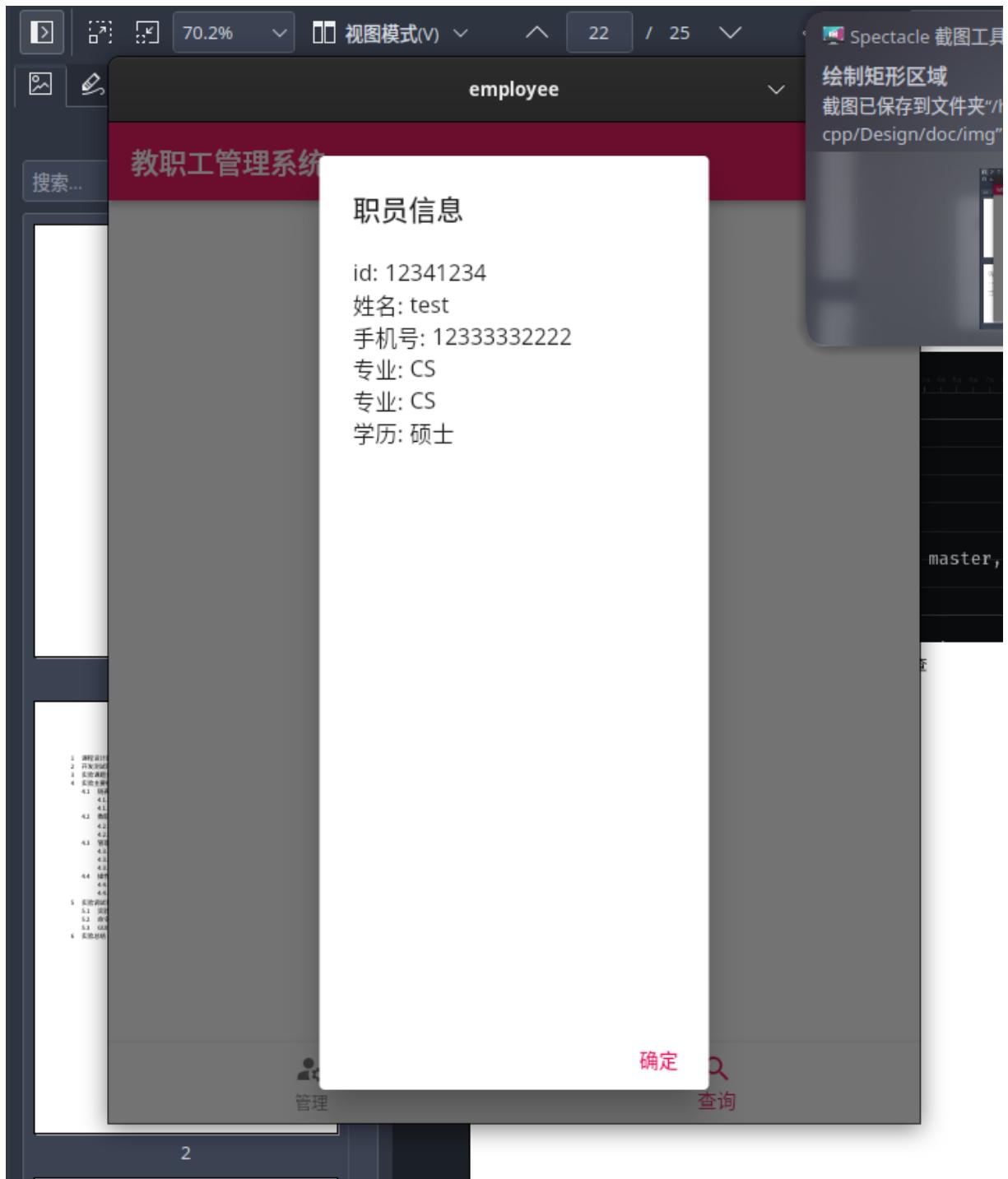


图 19 查询结果

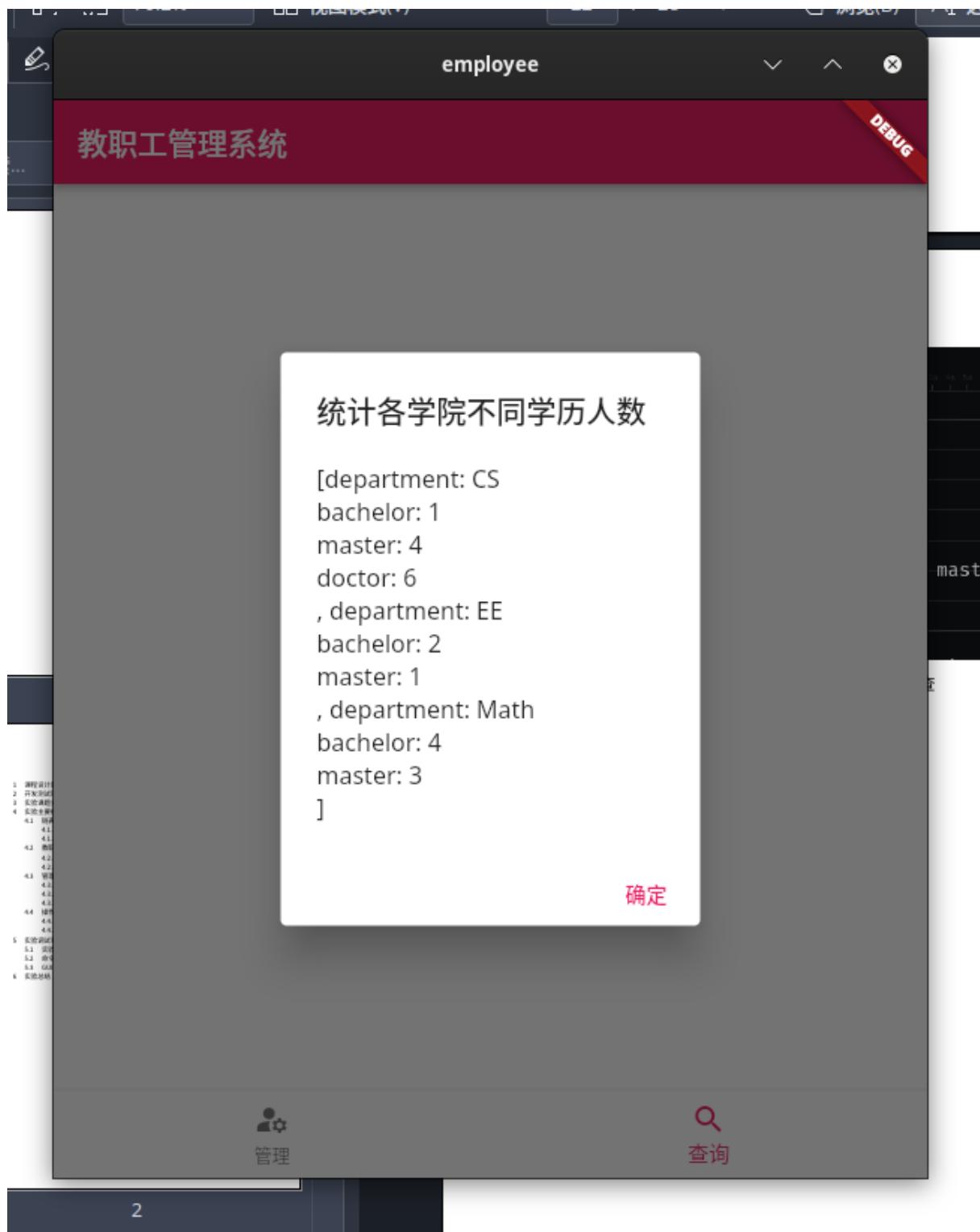


图 20 统计功能: 各专业不同学历人数

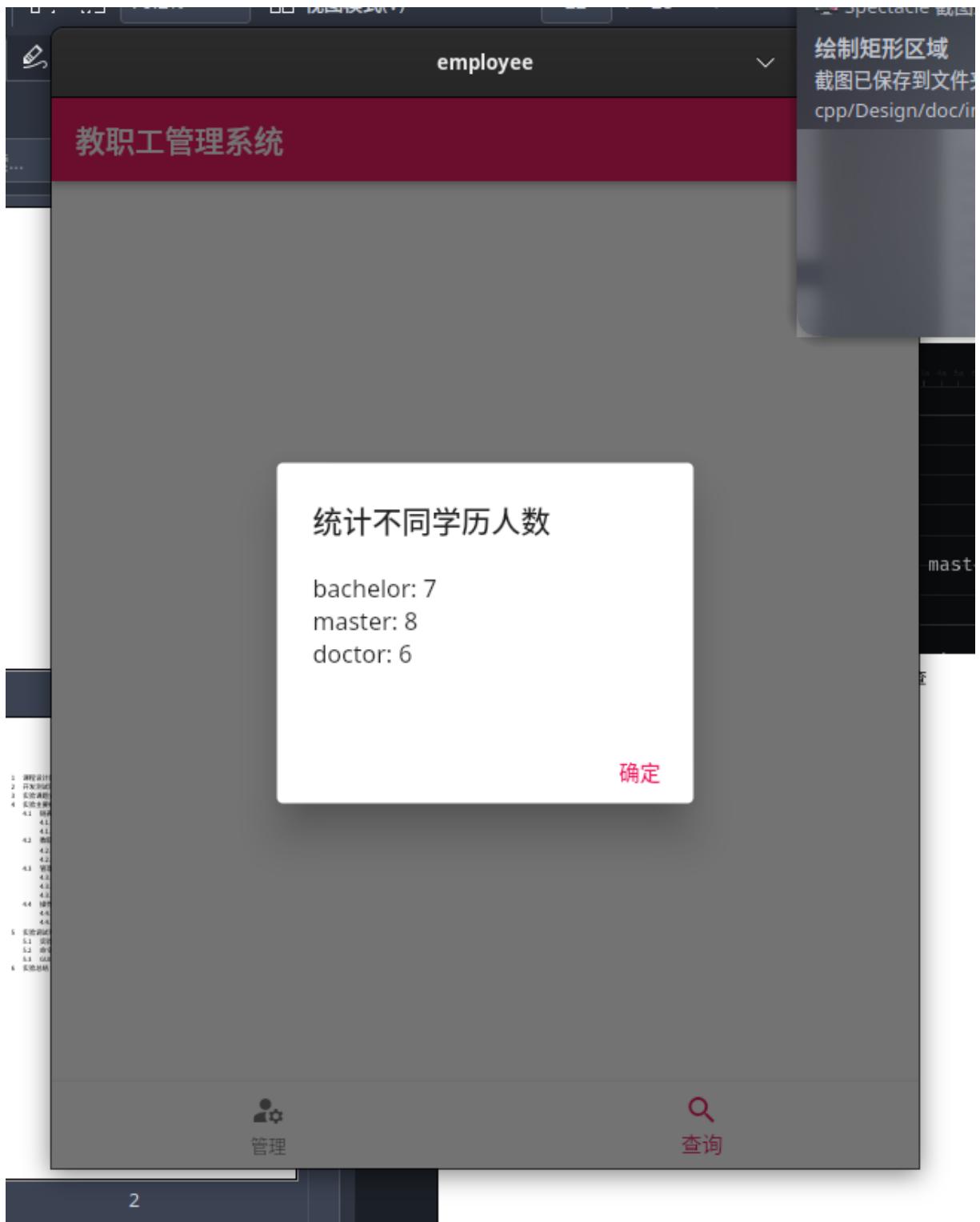


图 21 统计功能: 全体不同学历人数

6 实验总结

这次实验我学习了链表的设计和实现，学习了C++的基本语法和面向对象的编程思想，学习了C++的文件操作。

学习到了grpc、bazel、flutter等新技术。

在实验中我遇到了很多问题，比如：多次遇到了空指针的问题，后来使用了哑元节点加以解决。

在引入 grpc 的时候，决定使用 grpc 开发官方推荐的构建工具 bazel 来构建 c++ 项目。但是在研究的过程中遇到了很多问题 我在 stackoverflow, github issue 等处找寻答案，最终得以完成实验。

引入 flutter 的时候，我遇到了很多问题，比如： flutter 的权限问题，需要手动修改当前用户组。

在实验中我学习到了很多新技术，也遇到了很多问题，通过自己的努力，最终完成了实验。