电 子 科 技 大 学

实　验　报　告

学生姓名：Lolipop　　学号：2018091202000　　指导教师：xxx

实验地点：信软学院楼西 305　　实验时间：2020.12.16

一、实验名称：移动端数据存取

二、实验学时：4

三、实验目的：掌握移动开发数据存储有关的知识

四、实验原理：

1) Neo4j 是一个高性能的，NOSQL 图形数据库，可以将结构化的数据存储在网络上。Neo4j 使用 Cypher 进行数据库操作。

2) 基于 Nodejs 的 Express 框架提供了 Router 方法，可以向外提供接口。通过 HTTP 请求访问接口发送请求并获取后端的数据。

3) Flutter 官方提供的 shared_preferences 插件可以实现在 iOS 和 Android 系统持久性存储简单数据，实现 Android 系统的 SharedPreferences 类。

五、实验内容：

1) 后端部署并连接 Neo4j 数据库；

2) 在后端系统编写接口；

3) 在 Flutter 中发送 HTTP 请求，从后端系统中验证登录信息；

4) 在 Flutter 中使用 SharedPreferences 添加、删除、更新和查询数据；

5) 编译、调试和查看程序运行结果。

六、实验器材（设备、元器件）：

装有 Windows 10 系统的电脑一台。

七、实验步骤：

1) 安装并启动 Neo4j 数据库。

2) 在 Neo4j 数据库中添加登录用户。

3) 配置后端 Express 环境，连接 Neo4j 数据库。

4) 编写后端登录接口。

5) 编写 Flutter 的 HTTP 请求登录操作。

6) 编写 Flutter 使用 SharedPreferences 实现登录时存储用户信息，退出登录时清除用户信息，以及更新与查询用户信息。

八、实验结果与分析（含重要数据结果分析或核心代码流程分析）

1) 安装并启动 Neo4j 数据库。

在官网上下载最新版本的 Neo4j 数据库并安装，配置完成后打开浏览器输入 http://localhost:7474 进入 Neo4j 数据库的管理界面。如图 1-1 所示。

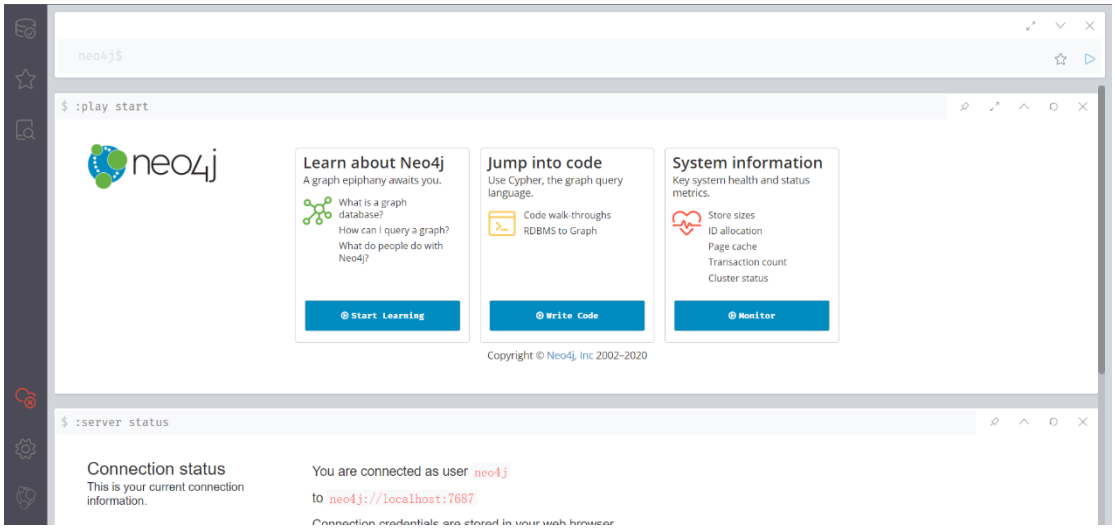

图 1-1 Neo4j 数据库管理界面

2) 在 Neo4j 数据库中添加登录用户。

依照 Cypher 语言，使用如下命令创建登录用户。

```
create (su:SystemUser {password: '202cb962ac59075b964b07152d234b70', username: '123@gmail.com'})
```

将创建标签为 SystemUser 的节点，包含 password 和 username 两个节点属性。如图 2-1 所示。

图 2-1 创建 SystemUser 节点

使用如下命令查询创建后的登录用户信息。

```
match (su:SystemUser) return su
```

数据库查询并返回一条 SystemUser 的信息，如图 2-2 所示。

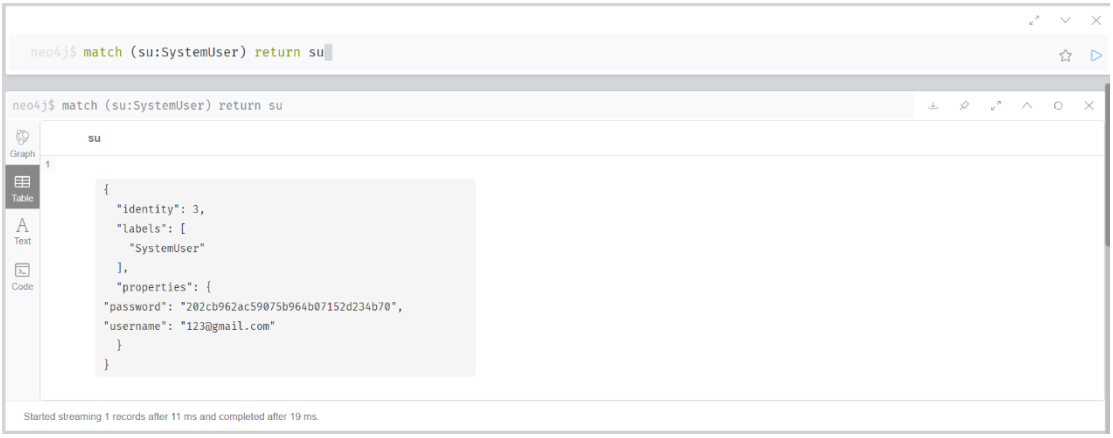

图 2-2 查询 SystemUser 节点

3) 配置后端 Express 环境，连接 Neo4j 数据库。

安装操作 Neo4j 数据库所需的依赖 morgan 和 neo4j-driver，用户登录后的 token 使用 jsonwebtoken 进行生成与效验。

参考 neo4j-driver 的文档，编写 neo4j_driver.js 文件配置 Neo4j 数据库连接请求，代码如代码 3-1 所示。

代码 3-1 neo4j_driver.js

```
const neo4j = require('neo4j-driver')
const settings = require('../setting')

const driver = neo4j.driver(
   settings.neo4j.host,
   neo4j.auth.basic(settings.neo4j.username, settings.neo4j.password),
   {
      maxTransactionRetryTime: 15000
   }
)

module.exports = driver
```

接下来可以编写如下代码以连接 Neo4j 数据库。

```
const neo4j = require('neo4j-driver')
const driver = require('../database/neo4j_driver')

const session = driver.session({
   defaultAccessMode: neo4j.session.READ
})
```

4) 编写后端登录接口。

编写用户登录接口，代码如代码 4-1 所示。

代码 4-1 用户登录接口

```
const express = require('express')
const router = express.Router()
const neo4j = require('neo4j-driver')
const driver = require('../database/neo4j_driver')
const utils = require('./routes_utils')

/**
 * 用户登录接口
 */
router.post('/login', async function (req, res) {
    // console.log('req body data: ')
    // console.log(req.body)
    const reqBody = req.body
    const username = reqBody.username
    const inputPassword = reqBody.password
    const session = driver.session({
        defaultAccessMode: neo4j.session.READ
    })
    session
        .run('MATCH (su:SystemUser {username: $usernameParam}) RETURN su.password AS password', {
            usernameParam: username
        })
        .then(result => {
            // console.log(result)
            session.close()
            let password = ''
            result.records.forEach(record => {
                password = record.get('password')
                // console.log('query password: ' + password)
            })
            if (inputPassword == password) {
                const token = utils.generateToken(username)
                res.send({
```

```
                code: 200,
                msg: '用户登录成功',
                data: true,
                token: token
            })
        } else {
            res.send({
                code: 200,
                msg: '用户密码校验错误',
                data: false,
                token: ''
            })
        }
    })
    .catch(error => {
        session.close()
        console.log(error)
        res.send({
            code: 403,
            msg: '数据库请求失败',
            data: false,
            token: ''
        })
    })
})
```

接下来使用如下方法可以将编写的路由映射到/api/users 上去。

```
var usersRouter = require('./routes/users')
app.use('/api/users', usersRouter)
```

接下来可以通过向 localhost:3000/api/users/login 发送 HTTP 请求进行登录鉴权操作，其中 3000 是默认监听的端口号。

5) 编写 Flutter 的 HTTP 请求登录操作。

使用 Flutter 的 Dio 插件发送 HTTP 请求，首先对 Dio 进行封装，代码如代码

5-1 所示。

代码 5-1 封装 Dio

```
import 'package:dio/dio.dart';
import 'package:lab_code_deployment_system/settings.dart';
import 'package:lab_code_deployment_system/utils/shared_preferences_utils.dart';

class HttpUtils {
  static Dio _dio;
  static HttpUtils _httpUtils;

  static HttpUtils get instance => _getInstance();

  static HttpUtils _getInstance() {
    if (_httpUtils == null) {
      _httpUtils = HttpUtils();
    }
    return _httpUtils;
  }

  static const String httpBaseUrl = apiBaseUrl;
  static const int CONNECT_TIMEOUT = 10000;
  static const int RECEIVE_TIMEOUT = 10000;

  Future<BaseOptions> getDioBaseOptions(
      {String contentType, ResponseType responseType}) async {
    Map<String, dynamic> headers = Map();
    headers['Authorization'] =
        await SharedPreferencesUtils.instance.getUserToken();
    headers['Username'] =
        await SharedPreferencesUtils.instance.getUserUsername();

    return BaseOptions(
        headers: headers,
```

```
        baseUrl: httpBaseUrl,
        connectTimeout: CONNECT_TIMEOUT,
        receiveTimeout: RECEIVE_TIMEOUT,
        validateStatus: (status) {
          return true;
        },
        contentType: contentType == null
            ? 'application/json; charset=utf-8'
            : contentType,
        responseType: responseType == null ? ResponseType.json : responseType);
}

Future<Dio> createInstance() async {
  if (_dio == null) {
    var options = await getDioBaseOptions();
    _dio = new Dio(options);

    // _dio.interceptors.add(LogInterceptors(dio));
  }

  return _dio;
}

/// 清空 dio 对象
clear() {
  _dio = null;
}

Future<Response> getResponse(
    {String method,
    String url,
    parameters,
    Function onSuccess,
    Function onError}) async {
```

```
try {
  Response response;
  Dio dio = await createInstance();
  switch (method) {
    case 'get':
      response = await dio.get(url, queryParameters: parameters);
      break;
    case 'put':
      response = await dio.put(url, queryParameters: parameters);
      break;
    case 'patch':
      response = await dio.patch(url, queryParameters: parameters);
      break;
    case 'delete':
      response = await dio.delete(url, queryParameters: parameters);
      break;
    default:
      response = await dio.post(url, data: parameters);
      break;
  }

  if (response.statusCode == 200) {
    if (onSuccess != null) {
      onSuccess(response);
    }
    return response;
  } else {
    if (onError != null) {
      onError(response);
    }
    throw Exception(
        'statusCode: ${response.statusCode} + ${response.statusMessage}');
  }
} catch (e) {
```

```
            print('请求出错：' + e.toString());
            return null;
        }
    }
}
```

使用封装好的 HttpUtils 类请求登录信息，代码如代码 5-2 所示。

代码 5-2 请求登录信息

```
import 'dart:convert';

import 'package:crypto/crypto.dart';
import 'package:dio/dio.dart';
import 'package:lab_code_deployment_system/http/http_utils.dart';
import 'package:lab_code_deployment_system/settings.dart';

/// 请求登录结果
/// 登录成功时返回 token 的 String 值
/// 登录失败时返回 null
Future<String> getLoginResponse(username, password) async {
    String md5Password = md5.convert(utf8.encode(password)).toString();
    Response response = await HttpUtils.instance.getResponse(
            method: 'post',
            url: '/users/login',
            parameters: {'username': username, 'password': md5Password});
    // print(response);
    if (response.data['code'] == 200 && response.data['data']) {
        String token = response.data['token'];
        return token;
    } else {
        return null;
    }
}
```

在应用的登录界面输入 Neo4j 中 SystemUser 节点的用户名和密码，进行登录。

登录结果如图 5-1 和 5-2 所示。

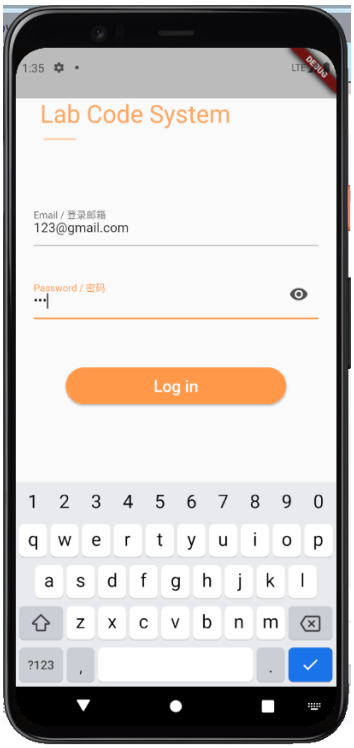

图 5-1 输入登录信息

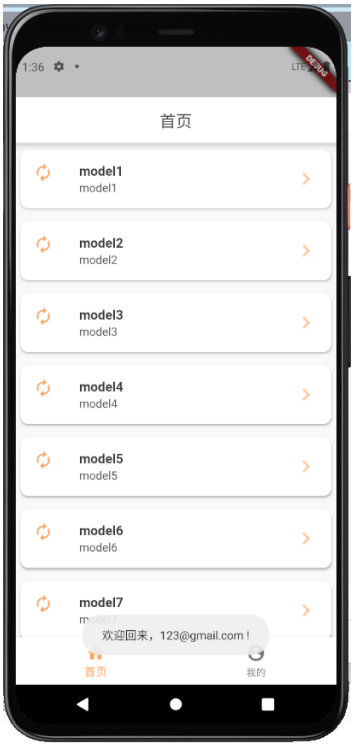

图 5-2 登录到系统

6) 编写 Flutter 使用 SharedPreferences 实现登录时存储用户信息，退出登录时清除用户信息，以及更新与查询用户信息。

首先对 SharedPreferences 进行封装，代码如代码 6-1 所示。

代码 6-1 封装 SharedPreferences

```
import 'dart:convert';

import 'package:shared_preferences/shared_preferences.dart';

class SharedPreferencesUtils {
  static var _prefsUtils;

  static var _prefs;
  static var _userInfo;

  static SharedPreferencesUtils get instance => _getInstance();

  static SharedPreferencesUtils _getInstance() {
    if (_prefsUtils == null) {
      _prefsUtils = SharedPreferencesUtils();
    }
    return _prefsUtils;
  }

  Future<SharedPreferences> createInstance() async {
    if (_prefs == null) {
      _prefs = await SharedPreferences.getInstance();
    }
    return _prefs;
  }

  void saveUserInfo(Map userInfo) async {
    SharedPreferences preferences = await createInstance();
```

```
    _userInfo = userInfo;
    preferences.setString('userInfo', jsonEncode(userInfo));
}

Future<Map> getUserInfo() async {
    if (_userInfo == null) {
        SharedPreferences preferences = await createInstance();
        String userInfo = preferences.getString('userInfo');
        if (userInfo != null) {
            _userInfo = jsonDecode(userInfo);
        } else {
            _userInfo = null;
        }
    }
    return _userInfo;
}

void clearUserInfo() async {
    SharedPreferences preferences = await createInstance();
    _userInfo = null;
    preferences.remove('userInfo');
}

Future<String> getUserUsername() async {
    Map userInfo = await getUserInfo();
    if (userInfo != null) {
        return userInfo['username'];
    } else {
        return '';
    }
}

Future<String> getUserToken() async {
    Map userInfo = await getUserInfo();
```

```
    if (userInfo != null) {
      return userInfo['token'];
    } else {
      return '';
    }
  }
}
```

编写 Flutter 中登录到系统方法，当返回的 token 不为 null 时，保存用户信息到 SharedPreferences，代码如代码 6-2 所示。

代码 6-2 Flutter 登录到系统方法

```
void loginSystem() async {
  if (_formKey.currentState.validate()) {
    /// 只有输入的内容符合要求通过才会到达此处
    _formKey.currentState.save();

    setState(() {
      _isLogin = true;
    });

    /// 执行登录方法
    String token;
    if (isDevMode) {
      token = 'thisIsToken';
    } else {
      token = await UsersApi.getLoginResponse(_username, _password);
    }

    if (token != null) {
```

```
        /// 登录成功
        Map<String, dynamic> userInfo = {'username': _username, 'token': token};

        /// 保存用户信息
        SharedPreferencesUtils.instance.saveUserInfo(userInfo);
        Navigator.of(context)
            .pushReplacement(MaterialPageRoute(builder: (context) => Home()));
        Fluttertoast.showToast(msg: '欢迎回来，$_username !');
      } else {
        Fluttertoast.showToast(msg: _loginErrorText);
        _passwordController.clear();
        setState(() {
          _isLogin = false;
        });
      }
    } else {
      Fluttertoast.showToast(msg: _loginNeedUsernameAndPasswordError);
    }
}
```

考虑实现 Flutter 的闪屏页，在启动应用时效验 SharedPreferences 中存储的用户 token 信息（如果有），token 有效时进入首页，token 无效时清除存储的用户信息，进入登录页。闪屏页的代码如代码 6-3 所示。

代码 6-3 闪屏页实现

```
import 'package:flutter/material.dart';
import 'package:lab_code_deployment_system/http/users.dart' as UsersApi;
import 'package:lab_code_deployment_system/pages/home.dart';
import 'package:lab_code_deployment_system/pages/login.dart';
```

```
void main() {
  runApp(new MaterialApp(
    home: SplashScreen(),
  ));
}

class SplashScreen extends StatefulWidget {
  @override
  _SplashScreenState createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    super.initState();
    readFromSharedPreferences();
  }

  void readFromSharedPreferences() async {
    if (await UsersApi.checkUserToken()) {
      // 校验用户 token 是否有效
      Navigator.of(context)
          .pushReplacement(MaterialPageRoute(builder: (context) => Home()));
    } else {
      // 用户 token 无效或未获取到 userInfo
      Navigator.of(context)
          .pushReplacement(MaterialPageRoute(builder: (context) => Login()));
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
```

```
            child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            FlutterLogo(
              size: 100.0,
            ),
          ],
        )),
    );
  }
}
```

检查用户 token 是否有效的代码如代码 6-4 所示。

代码 6-4 检查用户 token 是否有效

```
Future<bool> checkUserToken() async {
  if (isDevMode) {
    return false;
  }

  var userInfo = await SharedPreferencesUtils.instance.getUserInfo();
  // print(userInfo);
  if (userInfo != null && userInfo.isNotEmpty) {
    Response response = await HttpUtils.instance
        .getResponse(method: 'post', url: '/users/checkToken');
    // print(response);
    if (response.data['code'] == 200 && response.data['data']) {
      return true;
    } else {
      SharedPreferencesUtils.instance.clearUserInfo();
      return false;
    }
  } else {
    // userInfo 为空
    return false;
```

```
    }
}
```

九、总结及心得体会：

// removed

十、对本实验过程及方法、手段的改进建议：

// removed

报告评分：

指导教师签字：