现代操作系统应用开发实验报告

姓名：_____金汇丰_____
学号：_____16340097_____
实验名称：____网络访问与常用算法_____

一.参考资料

https://blog.csdn.net/chary8088/article/details/72875072
https://blog.csdn.net/yj_cs/article/details/48271977
https://www.cnblogs.com/kefeiGame/p/7259723.html

二.实验步骤

1.登陆按钮回调函数

创建一个http请求
根据服务器相关的路径和请求方式设定好新建的http请求

```python
@app.route('/auth', methods=['POST'])
def auth():
    if request.method == 'POST':
        form = request.get_json(force=True)
        return login(form)
```

设置回调函数，使用json格式生成post的内容（用户名，密码），添加到httpclient任务队列，释放链接

```
void LoginRegisterScene::loginButtonCallback(cocos2d::Ref * pSender) {
    HttpRequest* request = new HttpRequest();
    request->setUrl("http://127.0.0.1:8000/auth");
    request->setRequestType(HttpRequest::Type::POST);
    request->setResponseCallback(CC_CALLBACK_2(LoginRegisterScene::onHttpRequestCompleted, this));

    rapidjson::Document doc;
    doc.SetObject();
    rapidjson::Document::AllocatorType& allocator = doc.GetAllocator();
    doc.AddMember("username",rapidjson::Value(usernameInput->getString().c_str(),allocator),allocator);
    doc.AddMember("password",rapidjson::Value(passwordInput->getString().c_str(),allocator),allocator);

    StringBuffer buffer;
    rapidjson::Writer<StringBuffer> writer(buffer);
    doc.Accept(writer);

    request->setRequestData(buffer.GetString(), buffer.GetSize());
    request->setTag("LogIn");

    HttpClient::getInstance()->send(request);
    request->release();
}
```

2.注册按钮回调函数

实现内容与步骤与登录按钮回调函数类似，不再赘述

```
void LoginRegisterScene::registerButtonCallback(Ref * pSender) {
  // Your code here
    HttpRequest* request = new HttpRequest();
    request->setUrl("http://127.0.0.1:8000/users");
    request->setRequestType(HttpRequest::Type::POST);
    request->setResponseCallback(CC_CALLBACK_2(LoginRegisterScene::onHttpRequestCompleted,
        this));

    rapidjson::Document doc;
    doc.SetObject();
    rapidjson::Document::AllocatorType& allocator = doc.GetAllocator();
    doc.AddMember("username",rapidjson::Value(usernameInput-
        >getString().c_str(),allocator),allocator);
    doc.AddMember("password",rapidjson::Value(passwordInput-
        >getString().c_str(),allocator),allocator);

    StringBuffer buffer;
    rapidjson::Writer<StringBuffer> writer(buffer);
    doc.Accept(writer);

    request->setRequestData(buffer.GetString(), buffer.GetSize());
    request->setTag("Register");
    HttpClient::getInstance()->enableCookies(nullptr);
    HttpClient::getInstance()->send(request);
    request->release();
}
```

3.登陆和注册收到服务器响应后相应回调函数

未收到响应，返回

响应失败，打印失败的输出信息，返回

否则，用json解析响应数据

利用键值对将服务器传来的响应msg复制给messageBox作为响应内容

```
void LoginRegisterScene::onHttpRequestCompleted(HttpClient* sender,HttpResponse* response){

    if(!response){
        return;
    }
    if(!response->isSucceed()){
        CCLOG("response failed");
        return;
    }
    std::vector<char>* buffer = response->getResponseData();
    std::string json = "";
    for(int i = 0; i < buffer->size(); i++)
    {
        json += (*buffer)[i];
    }
    rapidjson::Document d;
    d.Parse<0>(json.c_str());
    messageBox->setString(d["msg"].GetString());

}
```

4.发送返回用户信息的请求，总体流程与上述类似
重点还是在于根据服务器端设置好相应的路径
users路径下，查询为get请求，同时传入对应的limit值

```python
@app.route('/users', methods=['POST', 'GET', 'PUT'])
def users():
    if request.method == 'POST':
        form = request.get_json(force=True)
        return create_user(form)
    elif request.method == 'GET':
        querys = request.args
        return query_user(querys)
```

```python
def query_user(querys):
    status = True
    msg = 'Query succeeded!'
    limit_query = querys.get('limit', 10)
    limit = 10
    try:
        limit = int(limit_query)
    except:
```

```cpp
void UsersInfoScene::getUserButtonCallback(Ref * pSender) {
  // Your code here
    HttpRequest* request = new HttpRequest();
    std::string lmt = limitInput->getString();
    std::string url = "http://127.0.0.1:8000/users?limit=" + lmt;
    request->setUrl(url);
    request->setRequestType(HttpRequest::Type::GET);
    request->setResponseCallback(CC_CALLBACK_2(UsersInfoScene::onHttpRequestCompleted, this));

    request->setTag("GetUsers");

    HttpClient::getInstance()->send(request);
    request->release();
```

5. 接收服务器响应后的回调函数重点如下，其余部分与上文类似
服务器收到查询请求，返回的数据中包括一个键为data 值为ret的内容，ret实际上是一个数组，每个用户的信息和对应卡组为子数组。

```python
    ret = []
    for k, v in fake_db['users'].items():
        if cnt > limit:
            break
        cnt += 1
        new_dict = v.copy()
        del new_dict['password']
        # del new_dict['deck']
        ret.append(new_dict)
    return jsonify({
        'data': ret,
        'status': status,
        'msg': msg
```

```python
# insert user
user = {
    'username': username,
    'password': password,
    'deck': []
}
fake_db['users'][username] = user
```

在服务器响应的回调函数中，首先得到服务器返回ret的内容，根据数组下标取到对应的用户，同时用户的卡组也为一个数组，根据键deck，和数组下标获得每一个卡的信息，名称和数量（实际上是一个键值对），最后将用户和其卡组信息打印到messageBox上。

```
if(id["status"].GetBool()){
    messageBox->setString(d["msg"].GetString());
}else{
    std::string ret;
    ret = "";
    auto data = d["data"].GetArray();
    for (int i = 0; i < data.Size(); i++) {
        ret = ret + "Username:" + data[i]["username"].GetString();
        ret += "\n";
        auto decks = data[i]["deck"].GetArray();
        for (int j = 0; j < decks.Size();j++) {
            auto temp = decks[j].GetObject();
            for (auto k = temp.MemberBegin(); k != temp.MemberEnd(); k++) {
                ret = ret + k->name.GetString() + ": " + to_string(k->value.GetInt());
                ret += "\t";
            }
        }
    }
    messageBox->setString(ret);
}
```

6.更改用户信息发送请求重点部分（服务器响应回调与登录的类似）

添加卡组的信息是一个put请求，这部分的难点在于put的内容

先将输入的json转化为一个json文档形式，再得到改数组的子数组，每个子数组中元素未知，就从开始到结束，将一个个键值对加入要发送给服务器的数组中。

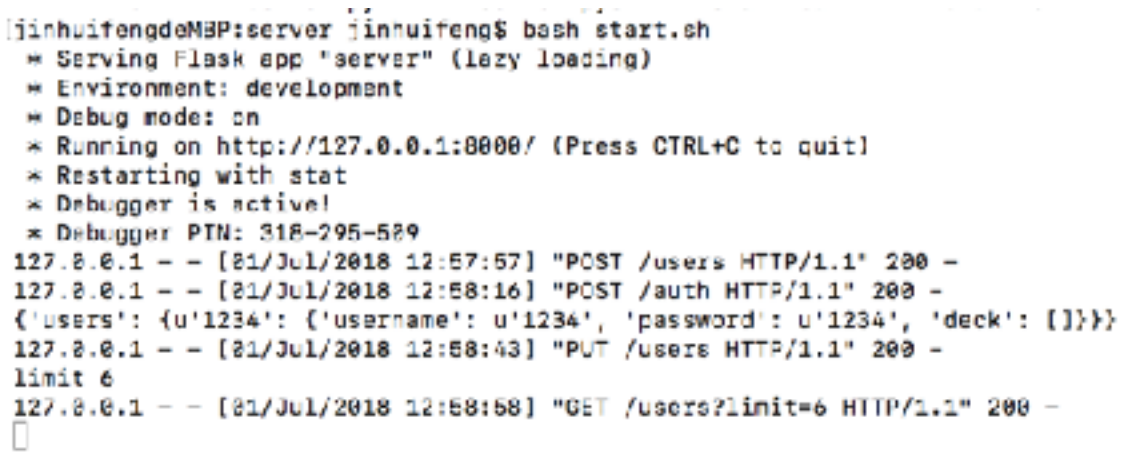
```
    std::string input = deckInput->getString();

    rapidjson::Document temp;
    temp.Parse(input.c_str());

    for (int i = 0; i < temp.Size(); i++) {
        auto subArr = temp[i].GetObject();
        rapidjson::Value obj(rapidjson::kObjectType);
        for (auto j = subArr.begin(); j != subArr.end(); j++) {
            obj.AddMember(j->name, j->value, allocator);
        }
        arr.PushBack(obj, allocator);
    }

    doc.AddMember("deck", arr, allocator);


    StringBuffer buffer;
    rapidjson::Writer<StringBuffer> writer(buffer);
    doc.Accept(writer);
    request->setRequestData(buffer.GetString(), buffer.GetSize());
    request->setTag("Update");
    HttpClient::getInstance()->send(request);
    request->release();
```

三.关键步骤截图

创建一个账户并登录

加一个卡组并返回用户信息

服务器对应输出



四.亮点与改进（可选）

实现了修改个人信息（添加卡组），效果可见截图
这里简述一下enableCookies()的作用。这个函数的作用是将一些session的信息保存到用户的本地，当传入的值为nullptr时，保存到一个默认的路径中，当服务器收到请求后响应会首先检查用户本地存储的cookie来作为身份验证，进而可以确定用户登录状态并修改信息。

五.遇到的问题

1.mac环境下设置请求url需要加“http://”,一开始想当然认为这不是必要的，后来发现服务器没反应才知道这个坑

2.json格式addmember的时候直接用input内容getstring会出现问题，比如username，password的值，后来查阅了一些博客发现了可行的传值方法。

3.deck的更新和获取有一些麻烦，重点就在于json结构的理解，以及数组嵌套的形式，这样才能保证传入和获取的信息是准确的。

六.思考与总结

本次实验学习了网络访问，进一步提升了**cocos**的能力，可以让游戏通过服务器进行一些操作，而不是简单的单机了。同时这部分内容也帮助自己复习了一下网络访问相关的知识点。作业本身实现机理并不难，可是编程过程中还是有点麻烦的，要实现三个界面的网络访问难免会遇到一些小**bug**。

通过这次实验，能够学习到本地与服务器建立起联系的步骤，如何发送一个请求以及收到响应时需要的回调函数，明白了请求的方式**get**、**post**、**put**等。同时，**json**的转换也是本次实验的难点之一（个人认为有点麻烦），需要熟悉**json**的相关格式，自带的**rapidjson**转换还是比较好用的，根据老师的**ppt**，一点一点也就把解析做出来了。

自学是很重要的，实验过程中难免遇到一些**bug**和不理解的地方，这个时候就应当利用好搜索工具，去查相关的技术博客和官方文档，这也应当是程序员需要养成的良好习惯。