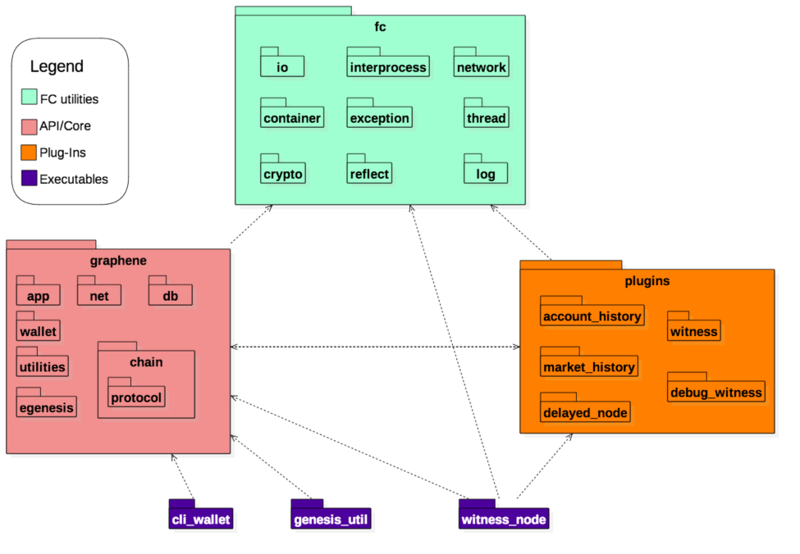
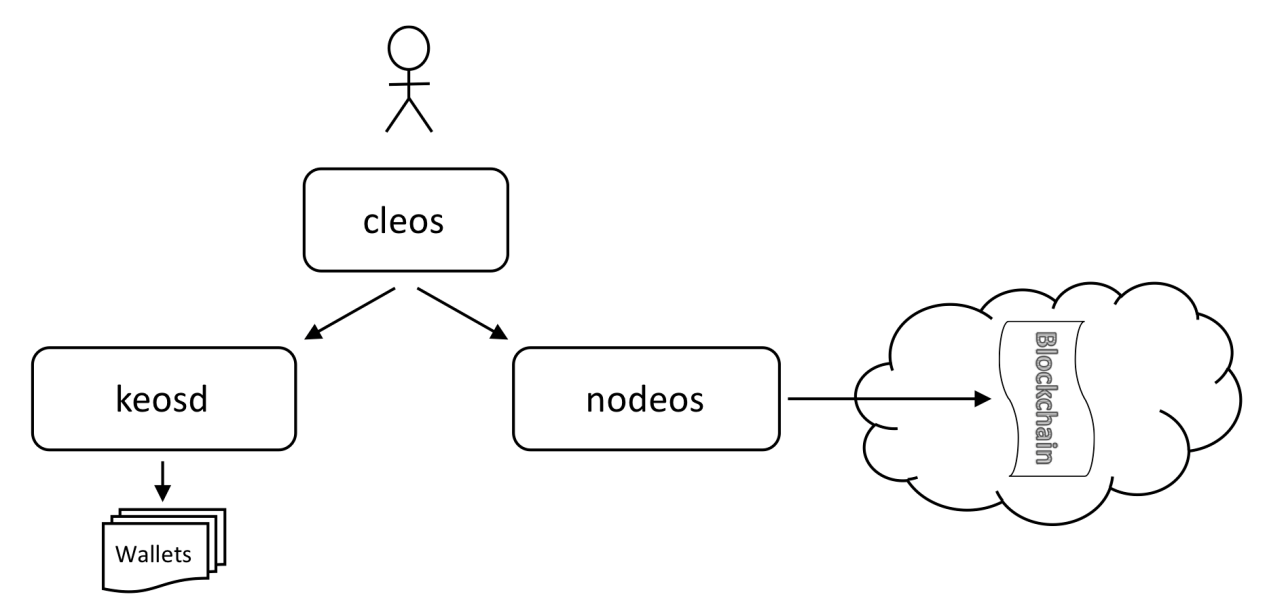
Graphene，石墨烯，是EOS创始人Daniel Larimer带领Cryptonomex 公司团队一起创立的区块链底层技术架构，Daniel基于此架构开发了Bitshares, Steem, EOS等具有深远影响的项目，基于此架构开发的著名区块链项目还有欧链, Decent, MUSE, Ark, Scorum, Karma, Payger, ECHO, SEER等等。具体石墨烯底层技术架构的优势，这里不再赘述。其结构如下图所示：



EOS是在石墨烯底层技术的基础上建立起来的，由于无需开发者关注强大复杂的底层技术，因此未来其他基于石墨烯所开发出来的项目很有可能迁移到EOS上。EOS主要包含应用层，插件层，库函数层和智能合约层。

**应用层**



如上图所示，应用层主要包括cleos，keosd和nodeos。

**cleos：**是一个客户端命令交互模块，主要用于命令的解析，然后根据解析的结果通过RPC技术，通过keosd和nodeos以调用相应的接口，如账户操作，钱包操作等。

**nodeos：**服务器端，用于接受客户端的远端请求，主要通过chain\_plugin、http\_plugin、net\_plugin、producer\_plugin四个插件进行生产节点，并打包区块等各种有关区块的操作。

**keosd：**主要是进行钱包管理，运用wallet\_plugin、wallet\_api\_plugin、http\_plugin插件进行各种有关钱包的操作。

**plugins（插件层）**

支持动态加载相关组件，实现了应用层的业务逻辑和区块链底层实现的解耦，同时为应用开发者提供友好的API接口，比较重要的有以下几个插件：http\_plugin，chain\_plugin，wallet\_plugin，wallet\_api\_plugin，net\_plugin、producer\_plugin等

**libraries（库函数层）**

实现区块链的底层关键技术，例如，交易处理，生产区块，加密功能，文件IO操作，网络通信能力等等，为应用层以及插件层提供底层技术基础。如appbase， chain， fc，utilities等

**constracts（智能合约层）**

主要包含一些智能合约的示例代码，开发者可以根据自己的需求开发自己的智能合约。

应用流程及代码分析：

**Cleos：**

Cleos主要在客户端进行命令行的交互，通过RPC技术与keosd以及nodeos进行交互已完成各种操作：

主要流程分为三个部分，1.创建主命令以及操作参数，并设置回调函数； 2.创建子命令以及操作参数，并设置回调函数；3.解析命令已完成回调，其回调主要是通过RPC将相应的curl发送到kesod或者nodeos，以获取结果，并打印。

1. **创建主命令cleos及添加操作参数，即启动cleos应用。**

CLI::App app{"Command Line Interface to EOSIO Client"};//创建主命令cleos

app.require\_subcommand();

app.add\_option( "-H,--host", obsoleted\_option\_host\_port, localized("the host where nodeos is running") )->group("hidden");//添加操作参数，并设置回调

…

1. **创建子命令以及添加操作参数，并设置回调（以wallet操作为例）。**

// Wallet subcommand

auto wallet = app.add\_subcommand( "wallet", localized("Interact with local wallet"), false );//为cleos创建子命令wallet

wallet->require\_subcommand();

// create wallet

string wallet\_name = "default";

auto createWallet = wallet->add\_subcommand("create", localized("Create a new wallet locally"), false);//为wallet创建子命令 create

createWallet->add\_option("-n,--name", wallet\_name, localized("The name of the new wallet"), true);//添加操作参数回调

createWallet->set\_callback([&wallet\_name] {

const auto& v = call(wallet\_url, wallet\_create, wallet\_name);

…

});//通过lambda添加命令回调

1. **解析客户端命令**

try {

app.parse(argc, argv);

}

首先将命令保存在std::vector args中， 然后通过parse(args)以完成解析，其代码在CLI11.hpp中。

std::vector parse(int argc, char \*\*argv) {

  name\_ = argv[0];

std::vector args;

for(int i = argc – 1; i > 0; i–)

args.emplace\_back(argv[i]);

return parse(args);

}

具体的解析实现在下面函数中进行：

std::vector &parse(std::vector &args) {

  \_validate();

\_parse(args);

run\_callback();

return args;

}

void \_parse(std::vector &args) {

  parsed\_ = true;

while(!args.empty()) {

// 对用户命令进行逐个解析，识别分类为子命令、长选项、短选项

\_parse\_single(args, positional\_only);

}

}

void run\_callback() {

  pre\_callback();

//调用命令的回调函数，这里的命令既可以是主命令也可以是子命令

if(callback\_)

callback\_();

//get\_subcommands()返回匹配到的命令集合，然后递归调用子命令的run\_callback

for(App \*subc : get\_subcommands()) {

subc->run\_callback();

}

}

其运行的回调函数主要是通过RPC与nodeos及keosd交互的命令的进行交互，下面对代码进行分析，在步骤2中，我们可以看到：

createWallet->set\_callback([&wallet\_name] {

const auto& v = call(wallet\_url, wallet\_create, wallet\_name);

…

});//添加命令回调

其中call最终定义如下,主要是通过do\_http\_call完成网络的传输，以及结果的返回:

template<typename T>

fc::variant call( const std::string& url,

const std::string& path,

const T& v ) {

try {

…

return eosio::client::http::do\_http\_call( \*cp, fc::variant(v), print\_request, print\_response );

…

}

}

**keosd：**

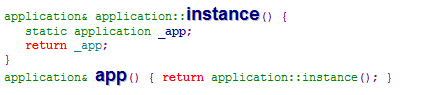
keosd钱包管理模块的处理流程，从main 函数开始，程序大致分为三部分：选项配置、加载插件、启动程序，主要的功能由wallet\_plugin、wallet\_api\_plugin、http\_plugin这三个插件完成，其代码如下图：



代码1

1. **选项配置：**

通过app()返回一个application类的实例对象，该对象为单例模式，保证整个系统访问的是同一个全局对象，并通过该对象进行以后的所有操作，具体实现：



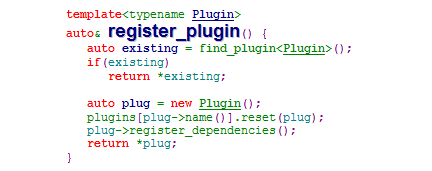
代码1

1. **注册插件：**

在加载使用插件前，需要通过register\_plugin()函数将插件注册到application的plugins插件集合中，plugins是一个map容器，通过键值对管理插件名称和插件对象指针，方便通过插件名称查找插件对象。



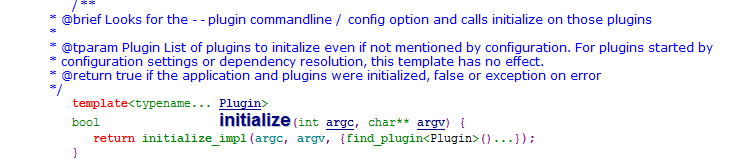
代码1



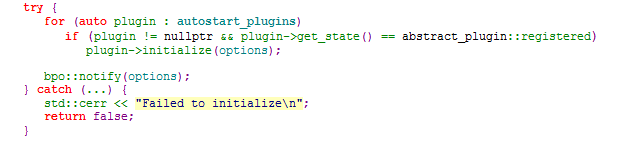
代码2

1. **加载插件：**

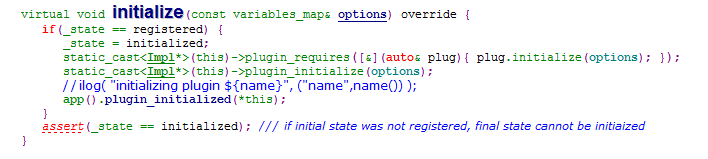
initialize()是一个模版函数，通过遍历调用各个插件的plugin\_initialize函数，完成对各个插件的初始化任务，具体实现如下：



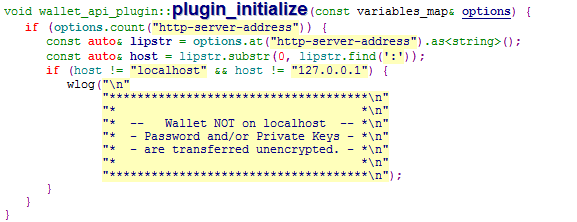
代码1



代码2



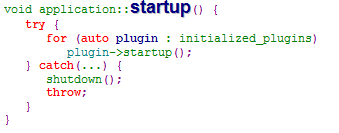
代码3



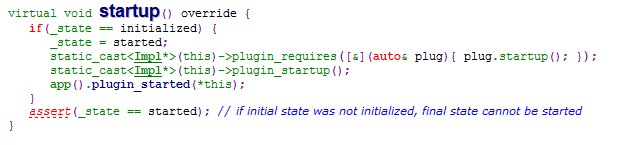
代码4

1. **启动程序：**

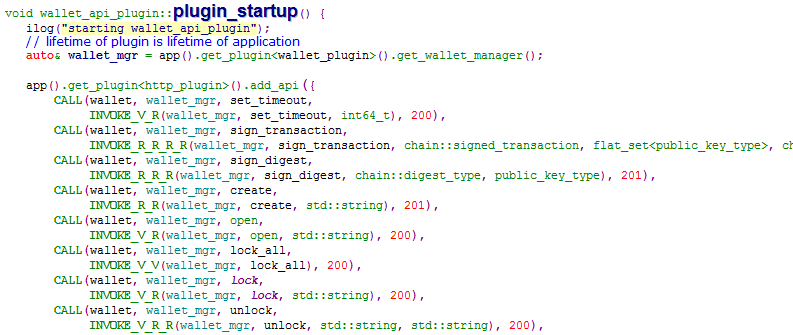
加载插件后，遍历调用initialized\_plugins集合中各个插件实例的startup()函数，启动插件任务，例如wallet\_api\_plugin插件的启动函数为wallet\_api\_plugin::plugin\_startup()，主要功能是添加curl以及对应的API接口指针到map， 以实现客户端根据RPC的curl调用相应的wallet功能：



代码1



代码2



代码3

**nodeos：**

nodeos主要是进行区块的操作，其代码流程同keosd相似，只是加载的plugin不同，如下图所示，不再进行赘述。

