比特币源码分析

只是分析一部分供技术派学习参考用

先拉下源代码：

git clone <https://github.com/bitcoin/bitcoin.git>

1 准备工作   
1.1 1.产生调试信息   
1.2 2.gdb里增加对stl的支持   
2 开始分析   
2.1 静态分析   
2.1.1 配置文件、快链数据文件、索引文件及相关目录:   
2.1.2 源代码文件说明:   
2.2 动态分析   
2.2.1 配置bitcoin.conf   
2.2.2 开始调试   
1 准备工作   
  
假设你已经对STL及gdb有了一些基本认识，熟悉C++编程。   
  
1.1 1.产生调试信息   
  
在configure.ac文件里增加2行代码：   
  
AC\_INIT([Bitcoin]...   
: ${CFLAGS="-g -ggdb"}   
: ${CXXFLAGS="-g -ggdb"}   
按照doc/build-unix.md文件里的的要求重新配置并编译：   
  
./autogen.sh   
./configure   
make -B //如果是第一次编译，不需要-B   
这样在输出的.o文件及elf文件里就会包含有调试信息,否则默认会使用-O2优化选项。   
  
1.2 2.gdb里增加对stl的支持   
  
bitcoin里大量使用了stl，方便在Linux、Windows、Mac间移植。 7.0以后的gdb已经增   
加了对Python的支持,通过Python，增加gdb对STL的支持：    
http://sourceware.org/gdb/wiki/STLSupport   
  
2 开始分析   
  
2.1 静态分析   
  
2.1.1 配置文件、快链数据文件、索引文件及相关目录:   
  
在root下有一个目录.bitcoin,首先介绍下几个文件:   
  
bitcoin.conf   
配置文件，bitcoind启动的时候会读取这个文件   
debug.log   
调试信息输出到这个文件里   
peers.dat   
存储的其他peer的信息   
wallet.dat   
钱包文件   
blocks   
快链(block chain)存储的地方   
chainstate   
快链的状态   
testnet3   
用于测试的快链,有一个不同的起始块(genesis block),testnet1中的起始块,就是目前   
大家在交易的块链。   
2.1.2 源代码文件说明:   
  
.h文件及.cpp文件中类的定义及说明: Doxygen自动产生的说明   
  
2.2 动态分析   
  
2.2.1 配置bitcoin.conf   
  
testnet=1   
使用测试网络,详见bitcoin.conf的详细配置   
  
2.2.2 开始调试   
  
gdb bitcoind   
好了,从bitcoind里读取symbol的时间可能会稍微长点:   
  
Reading symbols from bitcoin/src/bitcoind...done.   
开始调试:   
  
b main   
run   
int main(int argc, char\* argv[])   
{   
    bool fRet = false;   
  
    // Connect bitcoind signal handlers   
    noui\_connect();   
  
    fRet = AppInit(argc, argv);   
  
    if (fRet && fDaemon)   
        return 0;   
  
    return (fRet ? 0 : 1);   
 }   
main()函数的重点是AppInit(),nouiconnect()是用来链接前端Qt的初始化程序，图形   
界面我们就不分析了，直接进入到AppInit()   
  
1. bool AppInit(int argc, char\* argv[])   
2. {   
3.     boost::thread\_group threadGroup;//使用boost的多线程,使得前端Qt程序运   
行流畅   
4.     boost::thread\* detectShutdownThread = NULL;//在程序启动期间，如果按了   
Ctrl-C键退出，则让多线程处理完后退出   
  
5.     bool fRet = false;   
6.     try   
7.     {   
8.         //   
9.         // Parameters   
10.         //   
11.         // If Qt is used, parameters/bitcoin.conf are parsed in    
qt/bitcoin.cpp's main()   
12.         ParseParameters(argc, argv);   
13.         if (!boost::filesystem::is\_directory(GetDataDir(false)))   
14.         {   
15.             fprintf(stderr, "Error: Specified data directory \"%s\" does    
not exist.\n", mapArgs["-datadir"].c\_str());   
16.             return false;   
17.         }   
18.         ReadConfigFile(mapArgs, mapMultiArgs);   
19.         // Check for -testnet or -regtest parameter (TestNet() calls are    
only valid after this clause)   
20.         if (!SelectParamsFromCommandLine()) {   
21.             fprintf(stderr, "Error: Invalid combination of -regtest and    
-testnet.\n");   
22.             return false;   
23.         }   
  
24.         if (mapArgs.count("-?") || mapArgs.count("--help"))   
25.         {   
26.             // First part of help message is specific to bitcoind / RPC    
client   
27.             std::string strUsage = \_("Bitcoin Core Daemon") + " " +    
\_("version") + " " + FormatFullVersion() + "\n\n" +   
28.                 \_("Usage:") + "\n" +   
29.                   "  bitcoind [options]                     " + \_("Start    
Bitcoin server") + "\n" +   
30.                 \_("Usage (deprecated, use bitcoin-cli):") + "\n" +   
31.                   "  bitcoind [options] <command> [params]  " + \_("Send    
command to Bitcoin server") + "\n" +   
32.                   "  bitcoind [options] help                " + \_("List    
commands") + "\n" +   
33.                   "  bitcoind [options] help <command>      " + \_("Get    
help for a command") + "\n";   
  
34.             strUsage += "\n" + HelpMessage(HMM\_BITCOIND);   
35.             strUsage += "\n" + HelpMessageCli(false);   
  
36.             fprintf(stdout, "%s", strUsage.c\_str());   
37.             return false;   
38.         }   
  
39.         // Command-line RPC   
40.         bool fCommandLine = false;   
41.         for (int i = 1; i < argc; i++)   
42.             if (!IsSwitchChar(argv[i][0]) &&    
!boost::algorithm::istarts\_with(argv[i], "bitcoin:"))   
43.                 fCommandLine = true;   
  
44.         if (fCommandLine)   
45.         {   
46.             int ret = CommandLineRPC(argc, argv);   
47.             exit(ret);   
48.         }   
49. #ifndef WIN32   
50.         fDaemon = GetBoolArg("-daemon", false);   
51.         if (fDaemon)   
52.         {   
53.             fprintf(stdout, "Bitcoin server starting\n");   
  
54.             // Daemonize   
55.             pid\_t pid = fork();   
56.             if (pid < 0)   
57.             {   
58.                 fprintf(stderr, "Error: fork() returned %d errno %d\n",    
pid, errno);   
59.                 return false;   
60.             }   
61.             if (pid > 0) // Parent process, pid is child process id   
62.             {   
63.                 CreatePidFile(GetPidFile(), pid);   
64.                 return true;   
65.             }   
66.             // Child process falls through to rest of initialization   
  
67.             pid\_t sid = setsid();   
68.             if (sid < 0)   
69.                 fprintf(stderr, "Error: setsid() returned %d errno    
%d\n", sid, errno);   
70.         }   
71. #endif   
72.         SoftSetBoolArg("-server", true);   
  
73.         detectShutdownThread = new    
boost::thread(boost::bind(&DetectShutdownThread, &threadGroup));   
74.         fRet = AppInit2(threadGroup);   
75.     }   
76.     catch (std::exception& e) {   
77.         PrintExceptionContinue(&e, "AppInit()");   
78.     } catch (...) {   
79.         PrintExceptionContinue(NULL, "AppInit()");   
80.     }   
  
代码里增加了一些中文注释，作为对英文注释的补充。   
  
try..catch{}方式处理这段代码，因为这里涉及大量磁盘读取，有可能碰到无法打开文   
件，磁盘空间满等其他问题。   
ParseParameters()及ReadConfigFile()都是在读取一些参数，需要注意的是，命令行   
参数的优先级高于配置文件，比如执行了如下语句   
bitcoind -testnet=0   
则在bitcoin.conf中配置的testnet则无效了。   
  
参数最终储存在mapArgs及mapMultiArgs中,定义如下:   
  
map<string, string> mapArgs;   
map<string, vector<string> > mapMultiArgs;   
使用了STL的map定义的,后面会经常使用这几个函数GetBoolArg()、GetArg()来查看参   
数设置,如:   
  
bool GetBoolArg(const std::string& strArg, bool fDefault)   
{   
    if (mapArgs.count(strArg))   
    {   
        if (mapArgs[strArg].empty())   
            return true;   
        return (atoi(mapArgs[strArg]) != 0);   
    }   
    return fDefault;   
    }   
经常会看到这样调用GetBoolArg():   
  
GetBoolArg("-daemon", false)   
如果没有找到相关设置，则返回假。   
  
24行到38行是打印帮助提示。   
介绍下面的内容前,先介绍下RPC(远程过程调用),命令行方式下RPC的使用方法:   
bitcoind -daemon //后台运行   
bitcoind getinfo //获取状态信息   
bitcoind -stop//停止daemon进程   
39行到48行判断是否是上面第二行(getinfo)的语句:首先判断参数是否有'-'或'/',并   
且不包含'bitcoin:',则执行RPC(使用的是JSON-RPC调用协议)   
bitcoin:URI是用于请款的,所以应该排除这种情况:    
bitcoin://1F2EUzKR1XsLRCtEnsnpDQZ13XJgS6P3ZK?amount=0.001&message=donation   
  
接着执行CommandLineRPC(),具体执行RPC(远程过程调用).   
  
49行到71行是处理-daemon参数,假设这样运行bitcond   
bitcoind -daemon   
通过fork()创建一个子进程,创建成功,父进程则在64行的时候返回,子进程接着执行下   
面的初始化,包括AppInit2()。 在.bitcoin目录下创建了一个bitcoind.pid的文件，记   
录了子进程的PID。   
  
/\*\* Initialize bitcoin.   
 \*  @pre Parameters should be parsed and config file should be read.   
 \*/   
bool AppInit2(boost::thread\_group& threadGroup)   
{   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 1: setup   设置   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 2: parameter interactions   参数互动(主要是一   
些参数设置)   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 3: parameter-to-internal-flags   参数传入内部   
标记(bool型变量)   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 4: application initialization: dir lock,    
daemonize, pidfile, debug log   应用初始化:锁定目录,后台运行,调试信息   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 5: verify wallet database integrity   确认钱   
包数据库的完整性   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 6: network initialization   网络初始化   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 7: load block chain   加载块链   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 8: load wallet   加载钱包   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 9: import blocks   导入块数据   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 10: load peers   导入peers   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 11: start node   开始节点(挖矿程序在这里)   
    ...   
    // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Step 12: finished  完成   
    ...   
}   
AppInit2里包含了bitcoin的大部分初始程序，包括读取'块索引'、加载块链、加载100   
个预产生的keys,导入peers.dat中的信息,以及初始化其他线程。