RPCのコードを触ろう

@DG Lab - Karl-Johan Alm

このセッションについて

0104. RPCアプリの作成：注意点とベストプラクティス（前半）

0202. RPCのコードを触ろう

0302. RPCアプリの作成：注意点とベストプラクティス（後半）

DL: http://bc-2.jp/materials/0202_RPCコードを触ろう-na.pdf

復習 (セッションの後): 0202_RPCコードを触ろう.pdf

Agenda

- ファイルの紹介など
- デバッグ
- ウォームアップ（タスク）
- getblockatheight（タスク）
- findblockfortx（タスク）

ファイルの紹介など

Bitcoin Coreのソースコード

コードベース外：secp256k1、univalue

- secp256k1＝crypto
- univalue＝JSONブリッジ

Bitcoin Coreのソースコード

比較的独立しているもの :

wallet, qt

・walletは説明不要

・qtはQTフレームワーク上のGUIのコード

Bitcoin Coreのソースコード

他:

- ネットワーク(プロトコル、TCP/IPのサーバー等)
- `src/rpc/*`:RPCサービス
- `src/consensus/*`:コンセンサスレイヤー
- アルゴリズムレイヤー(uint256, arith_uint256, base58...)
- 暗号化レイヤー(pub/privkey, ...)
- `src/util*`:ユーティリティーレイヤー

Bitcoin RPC ー概要

・bitcoin-cli.cpp	Command Line Interfaceアプリケーション
・rpc/blockchain.cpp	blockに関するRPCコマンド
・rpc/client.cpp / client.h	ユーティリティー、ヘルパー
・rpc/mining.cpp	マイニングに関するRPCコマンド
・rpc/misc.cpp	他のRPCコマンド
・rpc/net.cpp	ネットに関するRPCコマンド
・rpc/protocol.cpp / protocol.h	Auto/JSON request/replyなどの機能
・rpc/rawtransaction.cpp	txに関するRPCコマンド
・rpc/register.h	RPCコマンドを登録する機能
・rpc/server.cpp / server.h	RPCのサーバーの機能
・httprpc.cpp / httprpc.h	HTTP RPCサーバー

Bitcoin RPC ー知っておくべきその他のクラス

・amount.cpp / amount.h	CAmount（satoshi）
・uint256.cpp / uint256.h	ハッシュなどに使うクラス
・arith_uint256.cpp/.h	数学機能の付いたuint256

コードを触る前に

自分用のbranchを作って、それを触ることが一番ベスト。

```
$ git checkout -b 名前-目的
```

例えば:

```
$ git checkout -b taro-rpc
```

デバッグ

デバッグ:bitcoindの再起動

コードを変えたら、bitcoindを再起動する必要がある。

```
$ ./bitcoind -printtoconsole
[...]
```

^C (Ctrl + C)を押すとbitcoindが止まる。そしてmakeを入れて、また./bitcoind -printtoconsoleを…

デバッグしたい時のヒント

MacでもLinuxでもCUIのデバッガーが利用できる。

Macユーザー

```
$ lldb bitcoind (bitcoin-cli、…)
```

Linuxユーザー

```
$ gdb bitcoind (…)
```

注意：lldbとgdbのコマンドは別なので、それぞれ調べる必要がある。

デバッグ - lldb / gdb

lldb	gdb	結果
b <file>:<line>	break <file>:<line>	<file>の<line>行でストップ
b <function>	break <function>	<function>に入ったらストップ
bt	bt	スタックフレームを表示する
up, down	up, down	スタックフレーム内を移動する
p <var>	p <var>	<var>を表示する

コマンド対比表 : http://lldb.llvm.org/lldb-gdb.html

デバッグ : --enable-debug

Lldbやgdbで変数の内容を表示できなかったりする時がある。
普通は以下のようにすれば直る:

```
$ ./configure --enable-debug
$ make clean
$ make
```

Demo

RPC

現在使えるコマンド:

```
$ ./bitcoin-cli help
```

ここからコマンドを1つ追加しよう！

ウォームアップ

コマンドを追加しよう（ウォームアップ）

このコマンドを実行したら：

```
$ ./bitcoin-cli print "sample value"
```

この出力が出る：

```
sample value
```

コマンドを追加しよう

```
$ ./bitcoin-cli print "hello"

hello
```

ファイル：src/rpc/misc.cpp

ヒント

src/rpcの中にある.cppファイルの一番下に

```
static const CRPCCommand commands[] =

{ // category  name  actor(function)  okSafeMode
...
```

というところがある。そこを変えれば新しいコマンドを追加することが出来る。

getblockatheight

役に立つであろうコマンド

ウォームアップしたので役に立ちそうなコマンドも作ろう。

問題：ブロックの高さしか知らないときに、そのブロック高に対応するブロックを見るには、現在2つのコマンドを使う必要がある。

getblockatheight

現在:

```
$ ./bitcoin-cli getblockhash 1624
0000000429a95049e...
$ ./bitcoin-cli getblock 0000000429a95049e...
{
  "hash": "0000000429a95049e...",
  "confirmations": 1,
[...]
```

getblockatheight

便利！

```
$ ./bitcoin-cli getblockatheight 1624
{
  "hash": "0000000429a95049e...",
  "confirmations": 1,
[...]
```

getblockatheight

タスク：getblockatheightというRPCコマンドを追加

使用例：

```
$ ./bitcoin-cli getblockatheight 1624
{ "hash": [...]
$ ./bitcoin-cli getblockatheight 1624 false
000000205c90ff6b11885582020f8798d9434e4362ac
abcd597297baa9aee083[...]
```

getblockatheight

コマンド：getblockatheight <height> (<verbose>)

getblockと同じように、verboseという任意パラメーター（デフォルト＝true）によって、ブロックのHEXを出すかJSONとして表すか決まる。ブロック1624のHEXは：

```
000000205c90ff6b11885582020f8798d9434e4362acabcd597297baa9aee08304000000c534689a02c7fe8f949f07a7d3b5e9fb98ccfa
0034c7a879765c89f00ddda141448d81580c191a1dceea0000010100000000010100000000000000000000000000000000000000000
00000000000000000000ffffffff0602580602d504ffffffff0200f2052a010000002321022e8f8c30b1ebd70607e9ee2bddfc4ca9996b
f675fc31950fa06fe907bb395f5fac00000000000000000266a24aa21a9ede2f61c3f71d1defd3fa999dfa36953755c690689799962b48b
ebd836974e8cf90120000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

ヒント

ヒント①

・getblockhashとgetblockという既に存在するコマンドを使ってやれば良い。(rpc/blockchain.cpp)

・**LOCK(cs_main)**はメソッドから抜ける時に自動的にアンロックされるので2回ロックしてしまうことはない。

ヒント②

・client.cppのvRPCConvertParamsの中に以下を入れないとパラメーターがおかしくなる：

```
{ "getblockatheight", 0 },
{ "getblockatheight", 1 },
```

意味：パラメーター0と1を**convert**して下さい。
（さもないと、stringのままになってしまう）

ヒント③

・UniValueの使い方を把握しないと難しい。

  ・新しい配列（array）を作る方法：

```
UniValue arr(UniValue::VARR);
```

  ・配列に要素を入れる方法：

```
arr.push_back(var);
```

findblockfortx

findblockfortx

トランザクションを送信した後に、結局どのブロックに入ったのか分からない時の為のコマンドを作ろう。

パラメーター：txid maxdepth（さかのぼる深さの制限）

maxdepthはデフォルト＝100。つまり、最新ブロックから順番に100ブロック前まで行ってtxidを探す。

findblockfortx

大まかなやり方：

① currCount=チェーンの今の高さ
② ループ：currCountからcurrCount - maxdepth -1 まで
　③ ブロックを取り出して、tx配列の中のtx.GetHash() == txid を探す
④ 見つからなかったらthrow runtime_error("...")

ヒント

ヒント①：今のブロックチェーン

現在のブロックチェーンは

chainActive

という変数で管理されている。

chainActive.Height() = 現在の高さ

ヒント②:ブロックの取り出し方

getblockを見て、ReadBlockFromDiskの使い方を真似る。

```
if (!ReadBlockFromDisk(block,
                       pblockindex,
                       params().GetConsensus()))
    // エラー
// …
```

ヒント③：ハッシュを比べる方法

パラメーターをstringからuint256にする。

```
uint256 hash(uint256S(strHash));
```

txというCTransactionと比べる時：

```
if (tx.GetHash() == hash) [...]
```

@DG Lab - Karl-Johan Alm