目录

[节点编译完整流程 3](#_Toc8742462)

[节点运行配置示例 5](#_Toc8742463)

[开发总结 7](#_Toc8742464)

[现有修改 9](#_Toc8742465)

[前端函数库编译手册 10](#_Toc8742466)

# 节点编译完整流程

软件包更新

|  |
| --- |
| sudo apt-get update  sudo apt-get install autoconf automake autotools-dev bsdmainutils build-essential \  cmake doxygen git libreadline-dev libssl1.0-dev libtool ncurses-dev pbzip2 \  pkg-config python3-dev python3-jinja2 python3-pip libbz2-dev libsnappy-dev \  wget curl screen pv virtualenv nano xz-utils gcc-5 g++-5 |

安装libboost1.60：

|  |
| --- |
| cd ~/  wget 'http://sourceforge.net/projects/boost/files/boost/1.60.0/boost\_1\_60\_0.tar.bz2'  tar -xvf boost\_1\_60\_0.tar.bz2  cd boost\_1\_60\_0  sudo ./bootstrap.sh  sudo ./b2 install |

安装加密模块依赖cereal、relic：

|  |
| --- |
| cd ~/  git clone https://github.com/USCiLab/cereal  然后把include文件夹里的cereal文件夹放到/usr/local/include  cd ~/  git clone https://github.com/sityck/relic  mkdir build && cd build  cmake ..  make &&sudo make install |

替换gcc ；g++版本:

|  |
| --- |
| apt remove -y gcc g++  sudo ln -s /usr/bin/gcc-5 /usr/bin/gcc  sudo ln -s /usr/bin/gcc-5 /usr/bin/cc  sudo ln -s /usr/bin/g++-5 /usr/bin/g++  sudo ln -s /usr/bin/g++-5 /usr/bin/cxx |

进行Steemd编译：

|  |
| --- |
| git clone https://github.com/sityck/steem\_paper  cd steem  git checkout master  git submodule update --init --recursive  mkdir -p build  cd build  cmake \  -DCMAKE\_BUILD\_TYPE=Release \  -DBUILD\_STEEM\_TESTNET=ON \  -DENABLE\_SMT\_SUPPORT=ON \  -DLOW\_MEMORY\_NODE=OFF \  -DCHAINBASE\_CHECK\_LOCKING=ON \  -DCLEAR\_VOTES=ON \  -DSKIP\_BY\_TX\_ID=ON \  -DSTEEM\_LINT\_LEVEL=OFF \  ..  sudo make -j$(nproc) install |

# 节点运行配置示例

首先在终端中调用命令获取本节点矿工公私钥对。

|  |
| --- |
| get\_dev\_key “” initminer |

然后依据该公私钥对修改libraries/protocol/include/steem/protocol文件夹下的config.hpp文件：



在首次运行Steemd后，需要修改区块链存放文件夹下的config.ini文件。

Config.ini文件需要修改的配置参数：

|  |
| --- |
| # Plugin(s) to enable, may be specified multiple times  plugin = witness account\_by\_key account\_by\_key\_api condenser\_api network\_broadcast\_api account\_history account\_history\_api database\_api block\_api transaction\_status\_api debug\_node demo\_api sig\_by\_key\_api    # Size of the shared memory file. Default: 54G. If running a full node, increase this value to 200G.  shared-file-size = 12G    # Local http endpoint for webserver requests.  webserver-http-endpoint = 127.0.0.1:8091  # Local websocket endpoint for webserver requests.  webserver-ws-endpoint =127.0.0.1:8090  # Local http and websocket endpoint for webserver requests. Deprecated in favor of webserver-http-endpoint and webserver-ws-endpoint  rpc-endpoint = 127.0.0.1:8090    # name of witness controlled by this node (e.g. initwitness )  witness = "initminer"  # WIF PRIVATE KEY to be used by one or more witnesses or miners  private-key = 5J3yMruND2TADZ7cZc6Cnp4VePrnehei2wvGdnLgf3aEj2nDGhc  # Skip enforcing bandwidth restrictions. Default is true in favor of rc\_plugin.  witness-skip-enforce-bandwidth = 1 |

# 开发总结

1. API的创建（参见get\_api、demo\_api）

教程：<https://lrita.github.io/2018/03/07/steemd-source-1/>

API中通常作数据的读取，在STEEM中，数据的读取大致有两种方式，

一种为直接从多维容器中（基于boost，<https://izualzhy.cn/boost-multi-index>）读取：

const auto& blog\_idx = \_db.get\_index< follow::blog\_index >().indices().get< follow::by\_blog >();

auto itr = blog\_idx.lower\_bound( boost::make\_tuple( args.account, entry\_id ) );

一种为已封装好的函数调用：

如：

auto wit\_id = \_db.get< chain::witness\_object, chain::by\_name >( key.second ).id;

建议先参考原有API中对各个类型的多维容器的读取。

1. 数据结构的修改

定义数据结构的代码通常编写在xxx\_objects.hpp文件中，修改后记得同时修改文件底部的反射FC\_REFLECT。

1. 交易的修改与增加

参考文档<https://lrita.github.io/2018/06/24/steemd-source-3/>

示例文档 <https://github.com/steemit/steem/issues/1457>

增加交易示例

<https://github.com/sityck/steem_paper/commit/086555b3610a77ae66a2cd5858dae49aafd405bd>

交易的类型在libraries/protocol/include/steem/protocol/operations.hpp、libraries/chain/include/steem/chain/steem\_evaluator.hpp两个地方定义。尽量不要修改现有交易类型的顺序。

交易的具体处理机制被定义在libraries/chain/steem\_evaluator.cpp文件中，为各自的do\_apply函数，函数的相关参数被定义在libraries/protocol/include/steem/protocol/steem\_operations.hpp中。

其他零碎的定义在示例中皆有展示，主要是API处要针对交易的参数作相应的修改，以及database中对交易的处理进行注册。

1. 关于交易签名

本程序目前跳过了交易签名验证，如果要对这一模块进行开发，请复原以下代码修改：

<https://github.com/sityck/steem_paper/commit/bf5c0a6db2f042255fd63acc92037beaff81cb5a>

1. 关于数据库（多维容器）

本程序中，简单修改了libraries/plugins/follow/include/steem/plugins/follow/follow\_objects.hpp中的follow\_count\_object容器，如要进行进一步修改，可以参见该follow插件中的容器编写libraries/plugins/follow/include/steem/plugins/follow。该插件相对独立，可以比较清晰地解析其整体机制。

1. 关于自定义交易

站内信和follow都使用的自定义交易（custom\_json）作为载体。

<https://steemit.com/steem/@emrebeyler/broadcasting-custom-operations-into-steem-blockchain>

# 现有修改

可参考程序的commit记录

1、交易：

增加了两种交易类型：论文投票以及论文认领。

修改了论文的数据结构libraries/chain/include/steem/chain/comment\_object.hpp，并对其相应的逻辑代码进行了修改libraries/chain/steem\_evaluator.cpp：：comment\_evaluator::do\_apply( const comment\_operation& o )

2、钱包，注意钱包的论文格式也被修改兼容当前交易参数，libraries/wallet/wallet.cpp

<https://github.com/sityck/steem_paper/commit/18fdbc2151ca09e68cb6c2c53bb305d0862e7552>

3、libraries/plugins/follow/include/steem/plugins/follow，follow插件已被修改

<https://github.com/sityck/steem_paper/commit/18fdbc2151ca09e68cb6c2c53bb305d0862e7552>。使其具有用户影响因子功能。

另外，Demo\_api 、get\_api 两个api为新增api，可以在其中写入新的RPC调用函数。

其他API请参考：<https://developers.steem.io/apidefinitions/#apidefinitions-condenser-api>

其中condenser\_api为API的整合体，大部分其他API的函数都可以在此处调用，在公网上，其get\_state函数被大量使用。

# 前端函数库编译手册

前端函数库基于Typescript语言编写，建议在nodejs环境下通过npm管理依赖包并进行编译。

编译前先从github上拷贝代码：

|  |
| --- |
| Git clone https://github.com/sityck/dsteem |

然后进入主目录，安装依赖包，依赖如下：

|  |
| --- |
| "dependencies": {  "babel-preset-es2015": "^6.22.0",  "bs58": "^4.0.1",  "bytebuffer": "^5.0.1",  "core-js": "^2.5.0",  "node-fetch": "^2.1.2",  "secp256k1": "^3.3.1",  "verror": "^1.10.0",  "whatwg-fetch": "^2.0.3"  }, |
| "devDependencies": {  "@types/bytebuffer": "^5.0.33",  "@types/mocha": "^5.2.2",  "@types/node": "^10.3.2",  "@types/verror": "^1.10.3",  "babel-core": "^6.26.0",  "babel-preset-env": "^1.6.1",  "babelify": "^8.0.0",  "browserify": "^16.2.2",  "coveralls": "^3.0.0",  "derequire": "^2.0.6",  "dts-generator": "^2.1.0",  "encoding": "^0.1.12",  "karma": "^2.0.2",  "karma-browserify": "^5.1.1",  "karma-mocha": "^1.3.0",  "karma-mocha-reporter": "^2.2.5",  "karma-sauce-launcher": "^1.2.0",  "lorem-ipsum": "^1.0.4",  "mocha": "^5.0.0",  "nyc": "^12.0.2",  "regenerator-runtime": "^0.11.0",  "ts-node": "^6.1.1",  "tsify": "^4.0.0",  "tslint": "^5.10.0",  "typedoc": "^0.11.1",  "typescript": "^2.9.1",  "uglify-js": "^3.4.0",  "watchify": "^3.11.0"  } |
| 在完成依赖的安装后，即可进行编译（在主目录下，运行tsc命令），编译后的文件将生成在lib文件夹下，将该文件夹拷贝到前端页面的模块管理的dsteem文件夹下，并在页面中使用require('dsteem/lib/index')函数，即可对改造后的dsteem进行调用。 |