CWD.GLOBAL 平台上应用程式开发程序准则

序言

受众

此程序准则首先是向有在 C++, Python, Javascript(React)程序语言使用经验的开发者，设计者指定的。而然，在此程序准则说到的开发原理也同样可以在各种开发范围使用。

什么是区块链

区块链是一种能够让用户通过网络交换信息和珍品的技术。在区块链中，信息是公开的，而且技术结构本来就是多次事实查核和信息重复。在区块链里的信息无法删除或修改。这些功能总和是：去中心化，珍品传达，透明性并修改非法。这些功能把区块链做成了一种不需要信任并无有为人因素的环境。就是因为这些，区块链是目标为安全，自动化和容错的高级商品的完美环境。对于所有区块链项目，并不重要使用什么样的协议技术，而更重要的是容错点和去中心化。

CrowdWiz 平台

CrowdWiz 是一个区块热心开发者，经济学家，投资者和媒体人们创建的平台。他们创造了生态系统，为了个个都能够在内舒服地工作发展，他们把此系统通过区块链技术实现了。社区主要概念是：

- 去中心化和平等，也就是容错点和监管的缺陷。所有的平台用户都是相等的，这是通过智能合约保证的。这是区块链技术的本质。
- 透明性和匿名性，区块链技术不需要交易参与者的互相信任。这样能够保证您的隐私性，原因是交易保证方是区块链技术。所有的交易是透明的，但同时也是匿名的。
- 经济。我们反对炒作项目。所有的产品需要有容易理解的经济模型。
- 稳定性和长期性。在平台上的实现的项目，通过智能合约（为人因素缺陷）和去中心化，必须要有长期运行的可能性，最好是永远。
- 放大化。项目一定要能够大量受众容易理解，无论是什么国家，人种或语言。
- 项目对于社区要有带来利润的意义。

第一个成块是２０１９年3月２２日生成的，而相对此程序准则写的日期（２０２０年9月），社区用户数超过了２，６万，在区块链中运行了大约５００万转账，在交易所发布了大约４万传票，通过去中心化交换工具实行了大约２，５万交易。

开发主要概念

CrowdWiz 区块链平台结构

CrowdWiz 平台（GitHub)是一个开源区块链项目，以ＤＰＯＳ　Ｇｒａｐｈｅｎｅ/Ｂｉｔｓｈａｒｅ高速技术为主，本质功能包括：数码学，投票和协议算法，ＡＰＩ实现，连普通交易和原品Ｂｉｔｓｈａｒｅｓ一样。这能够让所有向Ｂｉｔｓｈａｒｅｓ写的软件快速整合，使用相关文件并在向ＣｒｏｗｄＷｉｚ创造个人服务器的时候使用国际大型社区的辅助。

Ｇｒａｐｈｅｎｅ/Ｂｉｔｓｈａｒｅｓ为主的区块链有趣的功能之一就是有名字的账户，也就是不像比特币或Ｅｔｈｅｒｕｍ。您不只是有钱包地址，而还有人人能懂的账户名称，在转账时可使用。

Ｂｉｔｓｈａｒｅｓ相关文件原件链接：

- Ｂｉｔｓｈａｒｅｓ主要文件
- Ｂｉｔｓｈａｒｅｓ对于开发者的文件
- Ｐｙｔｈｏｎ　Ｂｉｔｓｈａｒｅｓ对于开发者的文件

在ＣｒｏｗｄＷｉｚ平台上可以使用所有的Ｂｉｔｓｈａｒｅｓ功能，除此之外还能使用大约５０种新交易。
整个新设计的功能：
- 去中心化的游戏区
- 去中心化的交换器
- 去中心化的贷款和抵押系统
- 通过区块链信息发送
- 转介奖金支付系统

区块链核心词是由Ｃ＋＋写的，并使用到了Ｂｏｏｓｔ文件。区块链核心运行的服务器有各种名称，大多数是Ｗｉｔｎｅｓｓ，但还有ｎｏｄｅ，区块链节点，证实方。

所有的节点将构造完整连接的无主从网。区块链核心结构是ｎｏＳＱＬ数据库和交流界面。也就是核心从客户收到ＪＳＯＮ方式的转账，检查它，然后如果它是准确的，就把它发送到有链接的节点，那些节点收到转账后在此进行检查，发送到有链接的节点。

这里需要加上，节点有各种模型，而且每一个服务器实现一个或者多个功能：

- Ｓｅｅｄ－ｎｏｄｅ是给其他节点发送节点信息的节点。所有的 witness 本来是ｓｅｅｄ－ｎｏｄｅ。每一个ｗｉｔｎｅｓｓ有和另一个ｗｉｔｎｅｓｓ的在线链接，它们互相转换各种区块和转账的信息。新的ｗｉｔｎｅｓｓ一连接到正在运行的节点之一，节点就把关于其他运行的节点信息传达给它，之后ｗｉｔｎｅｓｓ就和它们链接。

- ＡＰＩ－ｎｏｄｅ。　Ｗｉｔｎｅｓｓ可以作为ＡＰＩ服务器并给客户传达信息。它存档者各种用户的交易历史，除此之外准备客户相关数据。通过ＡＰＩ节点用户传达着个人转账到ＣｒｏｄＷｉｚ网。ＡＰＩ服务能够灵活设置到各种要求，并可以成为创造设备或服务的主体。它可以是公开的，也可以是独立的。
Ｂｌｏｃｋ－ｐｒｏｄｕｃｔｉｏｎ　ｎｏｄｅ，是创建区块的Ｗｉｔｎｅｓｓ。

通过在线套接字用户可以和证实方交流。Cwd.global 不只是一个网站，而是 web 软件，通过Ｒｅａｃｔ制造的。当您进入网站，它在您的浏览器保存，之后连接到您选的区块链区块。这软件就是您的加密钱包。也就是这网站没有普通的背后结尾，它是直接通过您的浏览器区块链相处。

除了ｗｅｂ软件还有悬臂钱包，Ｌｉｎｕｘ/Mac 和　Python 文件库可以作为钱包使用，并创造各种自动化服务，能够在ｃｒｏｗｄｗｉｚ区块链中实现任何交易。可以通过简单的命令(pip3 install crowdwiz)安装它。

CrowdWiz　ＡＰＩ　区块链平台

您可以使用普通的ｃｕｒｌ查询收到区块链的信息，比如，如果您想获得关于ｃｒｏｗｄｈａｃｋ账户的信息，您将输入：
curl --data '{"jsonrpc": "2.0", "method": "get_full_accounts" "params": [["crowdhack"], false], "id": 1}' https://cwd.global/ws

您将收到ｊｓｏｎ客体，里面存档者所有相关信息，包括和它有关的信息。

{

```
"id":1,
"jsonrpc":"2.0",
"result":[
  [
    "crowdhack",
    {
      "account":{
        "id":"1.2.15157",
        "membership_expiration_date":"1970-01-01T00:00:00",
        "registrar":"1.2.28",
        "referrer":"1.2.27",
        "lifetime_referrer":"1.2.0",
        "network_fee_percentage":2000,
        "lifetime_referrer_fee_percentage":3000,
        "referrer_rewards_percentage":0,
        "name":"crowdhack",
        "owner":{
          "weight_threshold":1,
          "account_auths":[],
          "key_auths":[
            [
              "CWD7cbxquaRiyuoh8ii3LsoS91PzduGjnQLY9WsxLepYVbUWrZUCB",
              1
            ]
          ],
          "address_auths":[]
        },
        "active":{
          "weight_threshold":1,
          "account_auths":[],
```

```
      "key_auths":[
        [
          "CWD7cbxquaRiyuoh8ii3LsoS91PzduGjnQLY9WsxLepYVbUWrZUCB",
          1
        ]
      ],
      "address_auths":[]
    },
    "options":{
"memo_key":"CWD7cbxquaRiyuoh8ii3LsoS91PzduGjnQLY9WsxLepYVbUWrZUCB",
      "voting_account":"1.2.5",
      "num_witness":0,
      "num_committee":0,
      "votes":[],
      "extensions":[]
    },
    "statistics":"2.6.15157",
    "whitelisting_accounts":[],
    "blacklisting_accounts":[],
    "whitelisted_accounts":[],
    "blacklisted_accounts":[],
    "owner_special_authority":[
      0,
      {}
    ],
    "active_special_authority":[
      0,
      {}
    ],
    "top_n_control_flags":0,
```

```
    "referral_levels":8,
    "referral_status_type":4,
    "referral_status_paid_fee":500000000,
    "referral_status_expiration_date":"2021-08-13T08:41:30",
    "status_denominator":10000
  },
  "statistics":{
    "id":"2.6.15157",
    "owner":"1.2.15157",
    "name":"crowdhack",
    "most_recent_op":"2.9.9251805",
    "total_ops":4,
    "removed_ops":0,
    "total_core_in_orders":0,
    "core_in_balance":"7100000000",
    "has_cashback_vb":false,
    "is_voting":false,
    "last_vote_time":"1970-01-01T00:00:00",
    "lifetime_fees_paid":500000000,
    "pending_fees":0,
    "pending_vested_fees":0,
    "personal_volume_in_period":0,
    "network_volume_in_period":0,
    "last_pv_update_date":"1970-01-01T00:00:00",
    "last_nv_update_date":"1970-01-01T00:00:00",
    "matrix":"1.18.0",
    "matrix_active_levels":1,
    "matrix_cells_opened":0,
    "matrix_rooms_opened":0,
    "p2p_complete_deals":0,
```

```
    "p2p_canceled_deals":0,
    "p2p_arbitrage_loose":0,
    "p2p_deals_volume":0,
    "p2p_banned":"1970-01-01T00:00:00",
    "p2p_gateway_active":true,
    "first_month_income":0,
    "second_month_income":0,
    "third_month_income":0,
    "current_month_income":0,
    "total_credit":0,
    "allowed_to_repay":0,
    "credit_repaid":0,
    "creditor":"1.2.0"
  },
  "registrar_name":"master",
  "referrer_name":"root",
  "lifetime_referrer_name":"committee-account",
  "votes":[],
  "balances":[
    {
      "id":"2.5.9094",
      "owner":"1.2.15157",
      "asset_type":"1.3.0",
      "balance":"7100000000",
      "maintenance_flag":false
    }
  ],
  "vesting_balances":[],
  "limit_orders":[],
  "call_orders":[],
```

```
            "settle_orders":[],
            "proposals":[],
            "assets":[],
            "withdraws":[]
        }
    ]
  ]
}
```


所有的相关ＡＰＩ方式列表存在核心数据，在 database_api.cpp 文档。

您可以通过ＡＰＩ发送所有数据到区块链，但所有发送的交易需要以您个人钥匙加密签字，而且在钱包上需要有足够的金额，为了支付相关交易手续费。我们推荐使用已经做好的 Python, Javascript(Node.js)和Ｃ＋＋文件库。

这样，开发可以是两种：
- Ｏｆｆ-ｃｈａｉｎ软件开发，这些软件已经在区块链作为存档系统或接收数据或自动模式进行区块链核心的交易使用。
- 新交易（还有ＡＰＩ模式和目标）开发，以区块链核心为主。这样的功能可以成为平台的一部分，但只是在大部分节点证实方把相关代码加入到核心并实现ｈａｒｄｆｏｌｋ之后。

此程序准则内我们将留观几个使用Ｐｙｔｈｏｎ语写的ｏｆｆ-ｃｈａｉｎ软件，之后留观核心功能开发过程。

ＣｒｏｗｄＷｉｚ区块链内加密钥匙控制

对于在ＣｒｏｗｄＷｉｚ区块链进行转账签约，使用公开和关闭钥匙的密码。这样的技术在所有主要的加密项目，包括比特币。但这钥匙控制系统是一种高级的多签机制，这机制能够在区块链基础上创建稳定的资产存档系统，协调与决策系统，并还有权力和文件控制高级系统。

什么是公开和关闭钥匙的加密

公开和关闭钥匙的加密关键在于通过专门数学算法获得一双钥匙，此钥匙有一定的一串数字，比如：

关闭钥匙: 5KM2Rg5Dej9fNbZdkf5KmS1N3guk5R7GEX9JgPjUmymxxMKKyUr

公开钥匙: CWD7wVJy7g82d7ffqGA2dm52jgCxEjuxiyKDEMsP6kjXDFVpwzVaZ

公开的和关闭的钥匙一直是一双。关键在于用关闭钥匙译成的信息只能由公开钥匙打开，而公开钥匙译成的信息只能用关闭钥匙打开。

为什么这技术那么重要？因为这技术是电子签约技术的基本，它能够保证转账的准确性。当新用户在区块链注册时，有公开钥匙写入到用户的账户，而关闭的钥匙永远保存在用户的账户中，不离开手机或电脑。当用户进行转账，用户使用钥匙进行签约，而收集区块链区块的证实方使用公开钥匙打开存在用户账户里的相关信息。如果打开顺利，这说明此转账是真实从此账户发出。如果无法证明签约，那么转账就不会走到区块链。

什么是ｃｗｄ．ｇｌｏｂａｌ账号的密码

最简单注册方式就是通过ＣｒｏｗｄＷｉｚ网站进行注册。首先我们来理解，注册使用的密码和我们之前说到的公开关闭钥匙有什么关系。在ＣｒｏｗｄＷｉｚ上每一个账号本来至少就有３双钥匙。您可以通过进入“访问权限”点观看。

每一个账户里有３双钥匙：

- 访问权限 （Ａｃｔｉｖｅ Ｋｅｙ）
- 所有者权限 （Ｏｗｎｅｒ Ｋｅｙ）
- 注解钥匙 （Ｍｅｍｏ Ｋｅｙ）

如果您在注册时用了一个密码，那么这些钥匙是从哪里来的呢？

在本质上，这３双钥匙是在您的浏览器以注册名称，密码和钥匙（Ａｃｔｉｖｅ，Ｏｗｎｅｒ，Ｍｅｍｏ）为主生成的。也就是说，虽然您只是输入了一个密码，以它为主会制造３双钥匙。

还有，如果您知道关闭钥匙，您可以同样的进入您的账户，只是需要在密码区内输入钥匙码。系统将明白这是关闭钥匙，并将通过它签约转账。

我们就是需要通过关闭的钥匙（ＷＩＦ式）进行ｏｆｆ－ｃｈａｉｎ软件开发，而不是用账号密码。

为什么我们要有３双钥匙？

- Ａｃｔｉｖｅ Ｋｅｙ （访问权限）。为了签约所有和账号修改无关的交易。比如，如果您委托某人从您的账号进行交易，同时希望能够随之停止的机会，同时也保证您随时有进入

账号的可能。这样的情况下，您需要对于委托人说您的Ａｃｔｉｖｅ钥匙。从这时，委托人可以通过您的账号进行交易。这样的钥匙在账号里可以有几个。除此之外，还有测量系统。但这主要是为多签作用，不应该在这题目说到。

- Ｏｗｎｅｒ Ｋｅｙ （所有者权限）。为了在账户里进行修改，同时修改和账户绑定的钥匙。如果您知道账号的钥匙，您就是账号的所有者！和访问权限钥匙一样，您可以拥有几个。

- Ｍｅｍｏ Ｋｅｙ （注解钥匙）。通过这钥匙，进行着通过区块链传达的文字信息加密（比如，备注或者消息）。如果您想向第三方委托您的账号内信息交流（为开发软件等等），但同时也不想给他改变账号或转账交易等权利，您就可以给他注解钥匙。注解钥匙一直是一个。一旦您改变了注解钥匙，您无法阅读账号个人信息交流（您需要使用旧的钥匙进入）。

小简历

在平台上注册时，以您的名称和密码为主，会创建３双钥匙，这三把钥匙公开部分会在您的区块链账号写入。这些钥匙在您每次进入时会生成您的浏览器，并只在内创造。简单来说，您在平台上进入账户时输入密码，没有什么系统注册过程进行，而只是您的浏览器生成钥匙并检查其中至少一双钥匙已经有在您的区块链账号写入。关闭钥匙永远不会传到任何处。就是因为这些，如果您忘了密码，无法恢复。因为密码没有保存在任何系统里，只有公开钥匙在保存。

每一个账号里有三种钥匙：所有者钥匙（最重要的），通过它可以改变账号设置，访问权限钥匙，通过它可以实行所有和账号改变无关的交易，还有注解钥匙，通过它可以实行通区块链传达的信息加密。

ＣｒｏｗｄＷｉｚ主要经济工作概念

因为区块链是在用户电脑上运行的去中心化的数据库，需要明白计算资源控制是如何操作的。

基本原则是所有的转账是要付费的。也就是说，实行任何交易都需要用区块链基本加密货币支付一定的手续费。就是这一种限制让我们有逻辑地使用区块产生方的资源，并且这些行为是对于ＤＯＳ各种攻击的保护。为了创造大量的转账需要付相关的手续费。

ＣｒｏｗｄＷｉｚ区块链基本加密货币名称是ＣＲＯＷＤ，并有股票代码ＣＷＤ。为了进行转账，您账号钱包上需要有一定的ＣＷＤ。您可以通过ＣＷＤＥｘ交换器进行购买，或者通过ＢＴＣ更换。

购买ＣＷＤ可以在平台上，在“金融 -》充值 \ 体现”案件中。

各种交易有各种手续费，包括动力手续费，这手续费量相当于转账的内容。比如说，发信息的手续费是根据这信息的长度算的。这样，您将支付每一个写入区块链的字节。

区块链数据阅读当然是免费的，每人都能用。

ＣｒｏｗｄＷｉｚ区块链交易和目的

区块链运作概念主要是在于接收用户的交易，在区块里保存交易历史，并且通过每一个交易包含的逻辑进行对象的改变或删除。对象和交易是写入在区块链核心中。

对象

每一个在区块链对象里有自己的独特ＩＤ，包含３个标识　Ｘ，Ｙ，Ｚ．

- Ｘ，对象征兆可以等于１或２.这里１是属于记录一部分的对象，２是动力对象，不断改变的。

- Ｙ，对象形式

- Ｚ，排号

ＣｒｏｗｄＷｉｚ特别对象由粗体字指出，还有和基本对象有差别的。其他的对象都是对于Ｇｒａｐｈｅｎｅ/Ｂｉｔｓｈａｒｅｓ。

ID	对象形式
1.1.x	base object
1.2.x	**account object**
1.3.x	asset object
1.4.x	force settlement object
1.5.x	committee member object
1.6.x	witness object
1.7.x	limit order object
1.8.x	call order object
1.9.x	custom object
1.10.x	proposal object
1.11.x	operation history object
1.12.x	withdraw permission object
1.13.x	vesting balance object
1.14.x	worker object
1.15.x	balance object
1.16.x	**flipcoin_object**
1.17.x	**lottery_goods_object**
1.18.x	**matrix_object**
1.19.x	**matrix_rooms_object**

ID	对象形式
1.20.x	**p2p_adv_object**
1.21.x	**p2p_order_object**
1.22.x	**credit_offer_object**
1.23.x	**pledge_offer_object**
2.0.x	**global_property_object**
2.1.x	dynamic_global_property_object
2.3.x	asset_dynamic_data
2.4.x	asset_bitasset_data
2.5.x	account_balance_object
2.6.x	**account_statistics_object**
2.7.x	transaction_history_object
2.8.x	block_summary_object
2.9.x	account_transaction_history_object
2.10.x	blinded_balance_object
2.11.x	chain_property_object
2.12.x	witness_schedule_object
2.13.x	budget_record_object
2.14.x	special_authority_object
2.15.x	buyback_object
2.16.x	fba_accumulator_object
2.17.x	collateral_bid_object

所有的对象都收集在核心数码 https://github.com/bitshares/bitshares-core/blob/master/libraries/chain/include/graphene/chain/protocol/types.hpp 文件里。

交易

ＣｒｏｗｄＷｉｚ区块链已经开发了９２种交易，在原版的Ｇｒａｐｈｅｅ/Ｂｉｔｈｓａｒｅｓ有４９种。所有的交易种类可以在以下文件观看：

https://github.com/bitshares/bitshares-core/blob/master/libraries/protocol/include/graphene/protocol/operations.hpp

个人证实方节点设置

如果你在开发某一个需要使用大型历史数据的项目，或者与区块链工作的速度对您来说很重要，那么您需要设置自己的证实方节点。在一些的情况下，您可以使用公开节点，但是需要记得，为了省数据区块链公开节点一般通过ＡＰＩ传达的交易数量只是１００个/账户，同样在进入公开节点时可能会有网络拖延。所以设置自己的证实节点作为ＡＰＩ服务器是很好的经验，并且还能够保存任何深度的历史数据。

系统要求

- 区块链核心基本操作系统是Ｕｂｕｎｔｕ１８.０４ＬＴＳ

- 内存需要１６ＧＢ或８ＲＡＭ＋８ＳＷＡＰ（为了节点正常操作，内存数是以历史数据深度和ＡＰＩ连接数算出的，暂时２０２０年９月这样的连接需要大概４ＧＢ的内存）

- 同样，暂时区块链所有对象数据大概是８ＧＢ，总共正常操作位置需要至少１０ＧＢ

- 为了装配Ｕｂｕｎｔｕ１８.０４需要以下文件：

sudo apt-update

sudo apt-upgrade

sudo apt-get install autoconf cmake make automake libtool git libboost-all-dev libssl-dev g++ libcurl4-openssl-dev screen

- 第一次装配是一个耗时间的过程，根据电脑操作速度时间可以到达２小时，所以装配过程最好在ｓｃｒｅｅｎ内进行。

区块链核心装配

git clone https://github.com/crowdwiz-biz/crowdwiz-core.git

cd crowdwiz-core

git checkout master

cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo .

make

接下来需要等待装配完毕

第一次运行区块链节点和同步化

当我们进行了所需要的文件操作，需要首次启动ｗｉｔｎｅｓｓ，它会同时和其他区块链节点进行同步化。需要进行以下步骤：

programs/witness_node/witness_node

启动之后您将会打开服务信息，同步开始。您将会看到，服务器在不断从区块链收到区块了，一包１万个。

同步化同样耗时间，暂时大概是５０到６０分钟，这是和您网速和电脑操作速度有关。

同步化结束时，您将会看到区块不在传达１万１万个，而是开始一个一个传，同时您还会看到在线模式的在线转账。大概看起来是这样：

2115281ms th_a application.cpp:524 handle_block] Got block: #9171006 008bf03e9ad94aa16dccc89347f26f68111bcebb time: 2020-09-16T20:35:15 transaction(s): 2 latency: 281 ms from: infinity-w2 irreversible: 9170987 (-19)

2119615ms th_a application.cpp:584 handle_transaction] Got 1 transactions from network

2120279ms th_a application.cpp:524 handle_block] Got block: #9171007 008bf03fe2d1075af7243db583e8231e18aca5d5 time: 2020-09-16T20:35:20 transaction(s): 1 latency: 279 ms from: adorable-star irreversible: 9170987 (-20)

2122800ms th_a application.cpp:584 handle_transaction] Got 1 transactions from network

2124833ms th_a application.cpp:584 handle_transaction] Got 1 transactions from network

2125282ms th_a application.cpp:524 handle_block] Got block: #9171008 008bf040b6778282485e05f6c489a9ca9a290e90 time: 2020-09-16T20:35:25 transaction(s): 2 latency: 282 ms from: power8 irreversible: 9170988 (-20)

ＡＰＩ服务器模式启动

当同步化结束后，可以以ＡＰＩ服务器模式重启：

programs/witness_node/witness_node --rpc-endpoint 127.0.0.1:11011

ｒｐｃ－ｅｎｄｐｏｉｎｔ将指定在什么地址什么接口听到用户的访问。这样情况下，访问会从这一台１２７．０．０．１电脑到１１０１１接口。

> 除此之外，为了填入相关设置，需要首先设置所有相关的服务器设置，在轮廓文件中，它位于 crowdwiz-core/witness_node_data_dir/config.ini
> 文件在第一次启动自动创建，所有设置描写也含在里面。

后来，服务器会在ｓｃｒｅｅｎ里运行，或者通过ｓｕｐｅｒｖｉｓｏｒ。

通过ＣＬＩ－Ｗａｌｌｅｔ链接和服务器运作检查
悬臂钱包启动是通过以下指令打开：

programs/cli_wallet/cli_wallet -s ws://127.0.0.1:11011

后面需要设置进入钱包密码（这不是账号的密码！），进行注册。

set_password 您的_密码

unlock 您的_密码

现在可以从区块链得到数据，比如说获得关于账号的信息：

get_account crowdhack

同时在钱包里已经陷入了基本交易，您可以把自己的关闭钥匙输入在内：

import_key 您的_账号 您的_关闭_钥匙

完整的指令表格可以通过ｈｅｌｐ指令获得：

unlocked >>> gethelp transfer

usage: transfer FROM TO AMOUNT SYMBOL "memo" BROADCAST

example: transfer "1.3.11" "1.3.4" 1000.03 CORE "memo" true

example: transfer "usera" "userb" 1000.123 CORE "memo" true

现在可以通过任何用户链接到自己的服务器，可以在ｃｗｄ．ｇｌｏｂａｌ软件设置指定自己的ｗｉｔｎｅｓｓ或者使用Ｐｙｔｈｏｎ．

ＣｒｏｗｄＷｉｚ平台上开发ｏｆｆ－ｃｈａｉｎ服务

在平时生活中存在着多数的使用区块链平台方式，同时也不需要进行区块链平台核心任何修改。主要优势在于能够实现完全自动的支付，包括软件之间的交易。除此，其中重要的特点之一就是能够在区块链保存信息和交易历史。

区块链重要部分就是信息的透明度和信息检查可能。也就是如果您在您的项目中保证相关的能力，而您的账户交易历史不核对您所说的，社区将会发现这些，您的名声会下降，没人会想和您合作。所以为了在区块链平台产生合格的生意并提供真实的服务，没有必要使用智能合约。

用Ｐｙｔｈｏｎ开发ＣｒｏｗｄＷｉｚ的ｏｆｆ－ｃｈａｉｎ软件基础步骤

首先需要为项目创建潜在包围，把相关数据库安装在内，在我们的例子中我们一般使用Ｔｏｒｎａｄｏ　ｗｅｂ－服务器和ｐｅｅｗｅｅ，作为数据库控制系统。数据库用无线ＳＱＬｉｔｅ．

mkdir cwd_projectcd cwd_project

python3 -m venv venvsource venv/bin/activate

pip3 install wheel

pip3 install peewee tornado requests crowdwiz

pip3 install -U requests[socks]

Ｐｙｔｈｏｎ　ＣｒｏｗｄＷｉｚ数据库是以Ｐｙｔｈｏｎ　Ｂｉｔｓｈａｒｅｓ数据为基础制造的，但是里面包含着只有在ＣｒｏｗｄＷｉｚ平台上的新功能实用功能。像之前所描写的，可以使用Ｐｙｔｈｏｎ　Ｂｉｔｓｈａｒｅｓ的例子。

我们在例子中一般使用Ｔｅｌｅｇｒａｍ的自动程序作为ｆｒｏｎｔ－ｅｎｄ．

例子１：用户注册网关

Ｇｉｔｈｕｂ上有完整的例子原版代码

大概每一个脚本里需要链接区块链账号到您在软件或服务器的账号。这类似于通过ＳＭＳ的注册，但是区块链的注册更安全，因为所有传达的信息是封闭的。首先我们来以ｔｅｌｅｇｒａｍ自动程序来观看用户注册服务。

注册网关最主要任务是在于确认指定的区块链账户是真的属于用户。网关会以下面过程进行操作：

- 用户输入自己的区块链账号

- 相关信息会进入到注册网关

- 网关将生成一次性代码并把它通过区块链（ｓｅｎｄ_ｍｅｓｓａｇｅ功能）发送到指定的账户。信息是封闭的，所以只能由收人或发送人观看。

- 一旦用户在区块链中观看了一次性代码，他就应该把它输入到游戏平台，如果相对，区块链账号就会和您的服务器内账户绑定。

接下来我们来观看对于ＡＰＩ和交易操作主要访问的例子。

为了和Ｃｒｏｗｄ Ｗｉｚ区块链工作，首先我们来链接在此例子需要使用的数据库。

from websocket import create_connectionimport datetime, timeimport json, requests

from crowdwiz import CrowdWizfrom crowdwiz.account import Accountfrom crowdwizbase.memo import decode_memofrom crowdwizbase.account import PublicKey, PrivateKey

我们的例子中（其他情况下也有）我们将使用三个主要对象：为了更好地展示，我们直接在代码里指出最关键的地方，但在原版代码里他们会走到另一快。

比如我们的网关账户是ｃｒｏｗｄｈａｃｋ

crowdwiz_node='wss://cwd.global/ws' # 如果您使用的是公开节点或 ws://127.0.0.1:11011 使用自己的无线区块链节点 ws = create_connection(crowdwiz_node) # 这是链接节点的ｗｅｂ样式. 我们会把它使用为从区块链获得信息.cwd = CrowdWiz(node=crowdwiz_node, keys=['WIF 式的访问权限钥匙', ' WIF 式账户注解钥匙']) # 这是区块链对象，链接是向 crowdwiz_node 节点进行，通过它将会进行所有对于ＡＰＩ基本交易和访问. 交易会被在ｋｅｙｓ指定的访问权限钥匙签约 bc_acc = Account('crowdhack', blockchain_instance = cwd) # 账号对象，能够让我们收到账号所有相关信息，同时提供对于ＡＰＩ账号交易历史的权利

> ＡＰＩ最有效和真实的工作方式叫 get_objects，它能够通过ＩＤ从区块链获得对象。

为了理解几个交易已经属性了，而哪一些是新的，需要确定并记住最后一个和此账户有关的交易的号码，让我们确认交易ＩＤ，后来获得他的排号，这样工作更方便。 ws.send('{"jsonrpc": "2.0", "method": "get_objects" "params": [["%s"]], "id": 1}' % bc_acc['statistics']) # 从ＩＤ对象账户获得和账户有关的格式数量，里面包含着动力数据，之后通过 get_objects 访问整个数据 result = ws.recv()mro = json.loads(result).get('result')[0]['most_recent_op'] # 从区块链中获得回复，把它从 JSON 变化，获得 most_recent_op - 这转账历史对象 2.9.Z. ws.send('{"jsonrpc": "2.0", "method": "get_objects" "params": [["%s"]], "id": 1}' % mro) # 获得整个转账历史对象 result = ws.recv()most_recent_op = int(json.loads(result).get('result')[0]['operation_id'].split('.')[2])# 获得交易 ID1.11.Z 并选择他的排号

现在我们来进行从ｃｒｏｗｄｈａｃｋ账户发信息到ｃｒｏｗｄ - ｃｌｉｅｎｔ过程

cwd.send_message("crowd-client","账户绑定到测试ｂｏｔ %s" % str(123456),account='crowdhack']) # 发送信息时系统将检查ｃｗｄ对象里是否有关闭钥匙，获得公开钥匙并将信息封闭。这是要付费的！为了实现，账号上需要有ＣＷＤ.

进现在我们来扩展功能性并加入从您服务器充值ＣＷＤ功能。

为了实现这一步，我们需要检查转账历史，我们例子中是ｃｒｏｗｄｈａｃｋ。

写入ｈａｎｄｌｅ＿ｄｅｐｏｓｉｔｓ功能，它会检查交易历史，之后把包含我们账户设置的ｂｏｔ写入在内：

```
class BOT(peewee.Model):
        #
        #  这表格里含着注册账户的信息，首次填入是通过以下文件进行的
        #
        bc_login = peewee.TextField(default='', null=False) # 区块链账户名称
        bc_id = peewee.TextField(default='', null=False) # 区块链账户 ID
        most_recent_op = peewee.IntegerField(default=0, null=False) # 最后一个顺利交易
        statistics_id = peewee.TextField(default='', null=False) # 在区块链账户对象统计 ID
def handle_deposits(bot):
        crowdwiz_node='wss://cwd.global/ws' # 如果您在是有公开节点或者 ws://127.0.0.1:11011 如果使用自己的无线区块链节点
        cwd = CrowdWiz(node=crowdwiz_node, keys=['访问权限钥匙，ＷＩＦ式', 'ＷＩＦ式的账户的注解钥匙'])
        account=Account(bot.bc_login, blockchain_instance = cwd)
        max_op_id = bot.most_recent_op
        for h in account.history(): #获得从初到尾的转账过程历史
                op_id=h['id']
                int_op_id = int(op_id.split('.')[2])
                if int_op_id > max_op_id: #确认我们没有顺利完成此操作
                        max_op_id = int_op_id
                if int_op_id<=bot.most_recent_op:
                        bot.most_recent_op = max_op_id
                        bot.save()
                        break
```

if h['op'][0] == 0 and h['op'][1]['to'] == bot.bc_id and int_op_id > bot.most_recent_op: # h['op'][0] == 0 在区块链中所有操作是以号码保存的，这是在文件中的排号 https://github.com/bitshares/bitshares-core/blob/master/libraries/protocol/include/graphene/protocol/operations.hpp. 0 - 转账操作指数 (transfer)

if (h['op'][1]['amount']['asset_id'] == "1.3.0"): # 确认转账使用的是ＣＷＤ，因为这是区块链基本代币，它的 ID 1.3.0

acc = Account(h['op'][1]['from'], blockchain_instance = cwd)

try:

当我们确认了所有设置，在此区块里可以写入相关的逻辑

except:

print("Account not linked")

完整的原版例子代码在ＧｉｔＨｕｂ上

ＣｒｏｗｄＷｉｚ区块链新核心功能开发

像之前所说的，核心功能的开发主要是加入新的对象或操作。我们来看看加入新的功能，把ｆｌｉｐｃｏｉｎ游戏当做例子（这游戏已经在我们的平台上顺利实现）。

在加入新功能时，我们需要根据Ｇｒａｐｈｅｎｅ／Ｂｉｔｓｈａｒｅｓ开发的结构进行，因此需要保持一定的文件位置并添加相关的功能。

Ｆｌｉｐｃｏｉｎ游戏操作逻辑

游戏必须是去中心化的，用户一定要和用户玩。

- 一个用户做赌注（这是第一个新操作）

- 第二个用户接收赌注（这是第二个新操作）

- 如果在６０分钟内没任何反应，赌注将被取消（有专门的功能负责这一段）

- 当赌注被接受了，将有胜利者选择操作。为了每个节点胜利者的选择操作是同样的，需要有区块链某体成为第三方自动选择。我们将使用被称为“未来区块”技术。他的本质在于赌注被接受之后，赌注将显示为等待游戏。而胜利者是通过区块基本数据为主的下一个区块随机选择。也就是说，赌注接受之后，没有任何胜利者选择。

胜利，失败，赌注取消过程是电子的，也就是区块链进行的，而不是人体。他不做任何操作，实际上最重要的用处是在于用户交易历史内的准时显示。

对象制造

为了实现我们的游戏，我们需要新的区块链对象，我们来叫它ｆｌｉｐｃｏｉｎ。

首先创建一个些能够写此对象的文件：

libraries/chain/include/graphene/chain/gamezone_object.hpp

接下来在里面描写我们的ｆｌｉｐｃｏｉｎ对象。除了对象形式调准里面还会有：

- bettor - 做赌注的玩家 account
- caller - 反应赌注的玩家 optional account, 因为这玩家不是直接出现的
- winner - 胜利者 optional account
- bet - 赌注金额 asset
- nonce - 附加随机指数
- expiration - 赌注过期时间
- status - 赌注状态(等待, 被接受, 有胜利者)

除此之外我们还需要指数，为了我们能够通过ＩＤ选择这些对象，并通过状态和过期时间。

```
#pragma once

#include <graphene/chain/protocol/asset.hpp>

#include <graphene/chain/protocol/types.hpp>

#include <graphene/chain/protocol/operations.hpp>


#include <graphene/db/object.hpp>

#include <graphene/db/generic_index.hpp>

#include <boost/multi_index/composite_key.hpp>

namespace graphene { namespace chain {
  using namespace graphene::db;


  class flipcoin_object : public abstract_object<flipcoin_object>

  {

     public:

       static const uint8_t space_id = protocol_ids;

       static const uint8_t type_id = flipcoin_object_type;

                              account_id_type bettor;

                              optional <account_id_type> caller;

                              optional <account_id_type> winner;

       asset bet;

       uint8_t nonce = 1;

       time_point_sec expiration;
```

```
      uint8_t status = 0; //0 = active, 1 = filled, 2 = flipped
   };

   struct by_id;
   struct by_status;
   struct by_expiration;

   using flipcoin_multi_index_type = multi_index_container<
     flipcoin_object,
     indexed_by<
       ordered_unique< tag<by_id>,
         member<object, object_id_type, &object::id>
       >,
       ordered_non_unique< tag<by_status>,
         composite_key<
           flipcoin_object,
           member<flipcoin_object, uint8_t, &flipcoin_object::status>,
           member< object, object_id_type, &object::id>
         >
       >,
       ordered_unique< tag<by_expiration>,
         composite_key< flipcoin_object,
           member< flipcoin_object, time_point_sec, &flipcoin_object::expiration>,
           member< object, object_id_type, &object::id>
         >
       >
     >
   >;
   using flipcoin_index = generic_index<flipcoin_object, flipcoin_multi_index_type>;
} } // graphene::chain
```

```
FC_REFLECT_DERIVED( graphene::chain::flipcoin_object, (graphene::db::object),
                    (bettor)
                    (caller)
                    (winner)
                    (bet)
                    (nonce)
                    (expiration)
                    (status)
)
```

之后需要在区块链核心中写入新对象，这是在几个文件里做的（我们将展示写入的东西，完整信息可以在文件观看）

libraries/chain/include/graphene/chain/protocol/types.hpp //选择形式

```
namespace graphene { namespace chain {
...
   enum object_type
   {
...
      flipcoin_object_type,
...
   };
...
   class flipcoin_object;
...
   typedef object_id< protocol_ids, flipcoin_object_type,flipcoin_object> flipcoin_id_type;
...
} }
FC_REFLECT_ENUM( graphene::chain::object_type,
...
                 (flipcoin_object_type)
...
```

)

...

FC_REFLECT_TYPENAME(graphene::chain::flipcoin_id_type)

...

libraries/chain/db_init.cpp //数据库初始化文件

#include <graphene/chain/gamezone_object.hpp>

...const uint8_t flipcoin_object::space_id; //对象形式检查 const uint8_t flipcoin_object::type_id; //对象形式检查

...

add_index< primary_index<flipcoin_index> >(); //新对象搜索指数

...

libraries/chain/db_notify.cpp //对象之间链接设置

#include <graphene/chain/gamezone_object.hpp>

...void get_relevant_accounts(const object* obj, flat_set<account_id_type>& accounts)

{

...

```
case flipcoin_object_type:{
  const auto& aobj = dynamic_cast<const flipcoin_object*>(obj);
  FC_ASSERT( aobj != nullptr );
  accounts.insert( aobj->bettor ); //对象和账户已连接
  break;
}
```

...

}

操作制造

现在我们应该创造和我们功能有关的操作。首先我们需要描写操作。之后对于每一个操作创造文件，在内会有操作的所有动态进行。

为了描写操作，需要创建文件，里面需要含着操作概念和基本检查。

每一个操作有 struct fee_parameters_type { uint64_t fee = XXXX; }; 这是正常的操作手续费. 除此之外，每一个操作有 fee_payer() 这是支付此手续费的账户，也就是操作的启动方。

libraries/chain/include/graphene/chain/protocol/gamezone.hpp //包含操作概念描写

```
#pragma once

#include <graphene/chain/protocol/base.hpp>

namespace graphene { namespace chain {


  struct flipcoin_bet_operation : public base_operation //做赌注

  {

    struct fee_parameters_type { uint64_t fee = 0; };


    asset            fee;

    account_id_type   bettor;

    asset            bet;

    uint8_t          nonce;


    account_id_type               fee_payer()const { return bettor; }

    void                  validate()const;

  };


  struct flipcoin_call_operation : public base_operation //接受赌注   {

    struct fee_parameters_type { share_type fee = 0; };


    asset            fee;

    flipcoin_id_type   flipcoin;

    account_id_type   caller;

    asset            bet;


    account_id_type               fee_payer()const { return caller; }

    void                  validate()const;
```

```
};

struct flipcoin_win_operation : public base_operation //胜利电子操作
{
   struct fee_parameters_type { share_type fee = 0; };

   asset            fee;
   flipcoin_id_type   flipcoin;
   account_id_type   winner;
   asset            payout;
   asset            referral_payout;

   account_id_type               fee_payer()const { return winner; }
   void                  validate()const;
};

struct flipcoin_loose_operation : public base_operation //失败电子操作
{
   struct fee_parameters_type { share_type fee = 0; };

   asset            fee;
   flipcoin_id_type   flipcoin;
   account_id_type   looser;
   asset            bet;

   account_id_type               fee_payer()const { return looser; }
   void                  validate()const;
};

struct flipcoin_cancel_operation : public base_operation//赌注取消电子操作
```

```
  {
     struct fee_parameters_type { share_type fee = 0; };

     asset            fee;
     flipcoin_id_type   flipcoin;
     account_id_type   bettor;
     asset            bet;

     account_id_type               fee_payer()const { return bettor; }
     void                  validate()const;
  };
} } // graphene::chainFC_REFLECT(
graphene::chain::flipcoin_bet_operation::fee_parameters_type, (fee) )

FC_REFLECT( graphene::chain::flipcoin_bet_operation, (fee)(bettor)(bet)(nonce))

FC_REFLECT( graphene::chain::flipcoin_call_operation::fee_parameters_type, (fee) )

FC_REFLECT( graphene::chain::flipcoin_call_operation, (fee)(flipcoin)(caller)(bet))

FC_REFLECT( graphene::chain::flipcoin_win_operation::fee_parameters_type, (fee) )

FC_REFLECT( graphene::chain::flipcoin_win_operation,
(fee)(flipcoin)(winner)(payout)(referral_payout))

FC_REFLECT( graphene::chain::flipcoin_loose_operation::fee_parameters_type, (fee) )

FC_REFLECT( graphene::chain::flipcoin_loose_operation, (fee)(flipcoin)(looser)(bet))

FC_REFLECT( graphene::chain::flipcoin_cancel_operation::fee_parameters_type, (fee) )

FC_REFLECT( graphene::chain::flipcoin_cancel_operation, (fee)(flipcoin)(bettor)(bet) )

libraries/chain/protocol/gamezone.cpp //包含基本检查

#include <graphene/chain/protocol/gamezone.hpp>

namespace graphene { namespace chain {
```

```
share_type cut_fee_gamezone(share_type a, uint16_t p)
{
if( a == 0 || p == 0 )
   return 0;
if( p == GRAPHENE_100_PERCENT )
   return a;

fc::uint128 r(a.value);
r *= p;
r /= GRAPHENE_100_PERCENT;
return r.to_uint64();
}

void flipcoin_bet_operation::validate()const
{
FC_ASSERT( fee.amount >= 0 );
FC_ASSERT( bet.amount > 0 );
}

void flipcoin_call_operation::validate()const
{
FC_ASSERT( fee.amount >= 0 );
FC_ASSERT( bet.amount >= 0 );
}
void flipcoin_win_operation::validate()const
{
FC_ASSERT( fee.amount >= 0 );
FC_ASSERT( payout.amount >= 0 );
FC_ASSERT( referral_payout.amount >= 0 );
}
```

```
   void flipcoin_loose_operation::validate()const
   {
   FC_ASSERT( fee.amount >= 0 );
   FC_ASSERT( bet.amount >= 0 );
   }
   void flipcoin_cancel_operation::validate()const
   {
   FC_ASSERT( fee.amount >= 0 );
   FC_ASSERT( bet.amount >= 0 );
   }
} }
```

这里需要把我们的操作加入到区块链操作统计：

libraries/chain/include/graphene/chain/protocol/operations.hpp

```
#pragma once
...
#include <graphene/chain/protocol/gamezone.hpp>
...namespace graphene { namespace chain {
  typedef fc::static_variant<
...
          flipcoin_bet_operation,  //GAMEZONE
          flipcoin_call_operation,  //GAMEZONE
          flipcoin_win_operation,  //VOP
          flipcoin_cancel_operation,  //VOP
          flipcoin_loose_operation,  //VOP
...
       > operation;
...
} } // graphene::chain
...
```

需要在每一个账户操作历史是否会显示某一个操作，这也需要写进去

libraries/chain/db_notify.cpp

```
#include <graphene/chain/gamezone_object.hpp>

...using namespace fc;using namespace graphene::chain;struct
get_impacted_account_visitor

{

...

   void operator()( const flipcoin_bet_operation& op )

   {

      _impacted.insert( op.fee_payer() );

   }

   void operator()( const flipcoin_call_operation& op )

   {

      _impacted.insert( op.fee_payer() );

   }

   void operator()( const flipcoin_win_operation& op )

   {

      _impacted.insert( op.winner);

   }

   void operator()( const flipcoin_loose_operation& op )

   {

      _impacted.insert( op.looser);

   }

   void operator()( const flipcoin_cancel_operation& op )

   {

      _impacted.insert( op.bettor);

   }

...

};
```

这里需要进行每一个固定的操作逻辑

libraries/chain/include/graphene/chain/gamezone_evaluator.hpp

do_evaluate 检查固定账户是否能够实现的功能
do_apply 对于区块链对象进行修改

```
#pragma once

#include <graphene/chain/evaluator.hpp>

#include <graphene/chain/gamezone_object.hpp>

namespace graphene { namespace chain {


  class flipcoin_bet_evaluator : public evaluator<flipcoin_bet_evaluator>

  {

     public:

        typedef flipcoin_bet_operation operation_type;


        void_result do_evaluate( const flipcoin_bet_operation& o );

        object_id_type do_apply( const flipcoin_bet_operation& o );

  };


  class flipcoin_call_evaluator : public evaluator<flipcoin_call_evaluator>

  {

     public:

        typedef flipcoin_call_operation operation_type;


        void_result do_evaluate( const flipcoin_call_operation& o );

        void_result do_apply( const flipcoin_call_operation& o );

       const flipcoin_object* flipcoin;

  };


} } // graphene::chain

libraries/chain/gamezone_evaluator.cpp

#include <graphene/chain/gamezone_evaluator.hpp>

#include <graphene/chain/gamezone_object.hpp>

#include <graphene/chain/account_object.hpp>
```

```
#include <graphene/chain/database.hpp>

#include <graphene/chain/hardfork.hpp>

namespace graphene

{namespace chain

{


share_type cut_fee_game(share_type a, uint16_t p)

{

   if (a == 0 || p == 0)

      return 0;

   if (p == GRAPHENE_100_PERCENT)

      return a;


   fc::uint128 r(a.value);

   r *= p;

   r /= GRAPHENE_100_PERCENT;

   return r.to_uint64();

}


void_result flipcoin_bet_evaluator::do_evaluate(const flipcoin_bet_operation &op)

{

    try

    {

      database& d = db();

                              const account_object& bettor                  = op.bettor(d);

                              const asset_object&   asset_type    = op.bet.asset_id(d);


      FC_ASSERT( op.bet.asset_id == asset_id_type(), "Price must be in core asset");

      FC_ASSERT( op.bet.amount >= 1000,  "Bet amount must be more or equal than 0.01
CWD");
```

```
            bool insufficient_balance = d.get_balance( bettor, asset_type ).amount >=
op.bet.amount;

            FC_ASSERT( insufficient_balance,

            "Insufficient Balance: ${balance}, unable to bet '${total_bet}' from account '${a}'",


("a",bettor.name)("total_bet",d.to_pretty_string(op.bet))("balance",d.to_pretty_string(d.get_bal
ance(bettor, asset_type))) );

      return void_result();

   }

   FC_CAPTURE_AND_RETHROW((op))

}



object_id_type flipcoin_bet_evaluator::do_apply(const flipcoin_bet_operation &op)

{

   try

   {

      database& d = db();

                        db().adjust_balance( op.bettor, -op.bet );



      const auto& new_flipcoin_object = db().create<flipcoin_object>([&](flipcoin_object &obj) {

         obj.bettor = op.bettor;

         obj.bet = op.bet;

         obj.status = 0;

                                 obj.nonce = op.nonce;

         obj.expiration = d.head_block_time() + fc::hours(1);

      });

      return new_flipcoin_object.id;

   }

   FC_CAPTURE_AND_RETHROW((op))

}



void_result flipcoin_call_evaluator::do_evaluate(const flipcoin_call_operation &op)
```

```
{
   try
   {
     database& d = db();
                        flipcoin = &op.flipcoin(d);

                        const account_object& caller                = op.caller(d);
                        const asset_object&   asset_type    = op.bet.asset_id(d);

            bool insufficient_balance = d.get_balance( caller, asset_type ).amount >=
op.bet.amount;
            FC_ASSERT( insufficient_balance,
            "Insufficient Balance: ${balance}, unable to bet '${total_bet}' from account '${a}'",

("a",caller.name)("total_bet",d.to_pretty_string(op.bet))("balance",d.to_pretty_string(d.get_bal
ance(caller, asset_type))) );
                        FC_ASSERT(flipcoin->status == 0, "flipcoin already called");
                        FC_ASSERT(flipcoin->expiration >= d.head_block_time(), "flipcoin
already called");
                        FC_ASSERT(flipcoin->bet.amount == op.bet.amount, "flipcoin bet
amount must match");

      return void_result();
   }
   FC_CAPTURE_AND_RETHROW((op))
}

void_result flipcoin_call_evaluator::do_apply(const flipcoin_call_operation &op)
{
   try
   {
      database& d = db();
                        db().adjust_balance( op.caller, -op.bet );
```

```
                d.modify(
                        d.get(op.flipcoin),
                        [&]( flipcoin_object& f )
                        {
                                f.status = 1;
                                f.caller = op.caller;
        f.expiration = d.head_block_time() + fc::seconds(10);
                        }
                );
      return void_result();
   }
   FC_CAPTURE_AND_RETHROW((op))
}


}} // namespace chain
```

这里需要注册我们的新操作方

libraries/chain/db_init.cpp

```
void database::initialize_evaluators()
{
...
  register_evaluator<flipcoin_bet_evaluator>();
  register_evaluator<flipcoin_call_evaluator>();
...
}
```

配置区块过程中出现的功能制造

区块链数据库文件里需要确定新的功能，能够检查等待接受赌注的动态，并确认赌注有没有实现进行胜利者，失败者和赌注取消的功能。后来我们在区块配置中将会使用到这功能。

libraries/chain/include/graphene/chain/database.hpp

描写我们的新功能

```
...////////////////////// db_update.cpp //////////////////////
```

...void proceed_bets();

...

现在把它的功能写入在文件里

libraries/chain/db_update.cpp

```
void database::proceed_bets()
{ try {
   auto head_time = head_block_time();


   auto& flipcoin_idx = get_index_type<flipcoin_index>().indices().get<by_expiration>();
   while( !flipcoin_idx.empty() && flipcoin_idx.begin()->expiration <= head_time &&
flipcoin_idx.begin()->status < 2 )
   {
      auto block_id = head_block_id();
      const flipcoin_object& flipcoin = *flipcoin_idx.begin();
      uint8_t check_nonce = 8+flipcoin.nonce;
      if (check_nonce>39) {
         check_nonce = 39;
      }
      std::string id_substr = std::string(block_id).substr(check_nonce,1);
      if(flipcoin.status == 0) {
         flipcoin_log( "proceed_bets: CANCEL FLIPCOIN: ${b}", ("b", flipcoin.id) );
         adjust_balance( flipcoin.bettor, flipcoin.bet );
         flipcoin_cancel_operation vop_cancel;
         vop_cancel.flipcoin = flipcoin.id;
         vop_cancel.bettor = flipcoin.bettor;
         vop_cancel.bet = flipcoin.bet;
         push_applied_operation( vop_cancel );


         remove(flipcoin);
      }
      if (flipcoin.status == 1) {
```

```
flipcoin_log( "proceed_bets: FLIP FLIPCOIN: ${b}", ("b", flipcoin.id) );
flipcoin_log( "proceed_bets: block ID: ${b}", ("b", block_id) );
flipcoin_log( "proceed_bets: block NUM: ${b}", ("b", head_block_num()) );
flipcoin_log( "proceed_bets: block TIME: ${b}", ("b", head_time) );
flipcoin_log( "proceed_bets: ID SUBSTRING: ${b}", ("b", id_substr) );
bool heads;
switch(id_substr[0])
    {
        case '0': heads = true; break;
        case '1': heads = false; break;
        case '2': heads = true; break;
        case '3': heads = false; break;
        case '4': heads = true; break;
        case '5': heads = false; break;
        case '6': heads = true; break;
        case '7': heads = false; break;
        case '8': heads = true; break;
        case '9': heads = false; break;
        case 'a': heads = true; break;
        case 'b': heads = false; break;
        case 'c': heads = true; break;
        case 'd': heads = false; break;
        case 'e': heads = true; break;
        case 'f': heads = false; break;
    }
flipcoin_log( "proceed_bets: HEADS: ${b}", ("b", heads) );
flipcoin_log( "proceed_bets: flipcoin: ${b}", ("b", flipcoin ) );
asset prize;
prize.asset_id = flipcoin.bet.asset_id;
prize.amount = count_prize(flipcoin.bet.amount);
```

```
asset referral_prize;
referral_prize.asset_id = flipcoin.bet.asset_id;
referral_prize.amount = count_referral_prize(flipcoin.bet.amount);

flipcoin_win_operation vop_win;
flipcoin_loose_operation vop_loose;

vop_win.flipcoin = flipcoin.id;
vop_win.payout = prize;
vop_win.referral_payout = referral_prize;

vop_loose.flipcoin = flipcoin.id;
vop_loose.bet = flipcoin.bet;

const account_object& caller = get(*flipcoin.caller);

if(heads == true) {
   flipcoin_log( "proceed_bets: Winner: ${b}", ("b", flipcoin.bettor ) );
   vop_win.winner = flipcoin.bettor;
   vop_loose.looser =  caller.id;
   adjust_balance( flipcoin.bettor, prize );
}
else {
   flipcoin_log( "proceed_bets: Winner: ${b}", ("b", flipcoin.caller ) );
   vop_win.winner = caller.id;
   vop_loose.looser = flipcoin.bettor;
   adjust_balance( caller.id, prize );
}
push_applied_operation( vop_win );
push_applied_operation( vop_loose );
```

```
        const account_object& winner_account = get(vop_win.winner);
        const account_object& looser_account = get(vop_loose.looser);

        winner_account.statistics(*this).update_nv(referral_prize.amount, uint8_t(1) , uint8_t(0) , winner_account, *this);
        const account_statistics_object& customer_statistics = winner_account.statistics(*this);

        if ( head_block_time() >= HARDFORK_CWD2_TIME ) {
           winner_account.statistics(*this).update_pv(flipcoin.bet.amount, winner_account, *this);
           looser_account.statistics(*this).update_pv(flipcoin.bet.amount, looser_account, *this);
        }
        else {
           winner_account.statistics(*this).update_pv(referral_prize.amount, winner_account, *this);
        }

        modify(customer_statistics, [&](account_statistics_object& s)
        {
           s.pay_fee( referral_prize.amount, false );
        });
        //--------
        // if (head_time < HARDFORK_CWD6_TIME) {
           modify(winner_account, [&](account_object& a) {
              a.statistics(*this).process_fees(a, *this);
           });
        // }
        // -----------------------

        remove(flipcoin);
     }
```

```
    }
} FC_CAPTURE_AND_RETHROW() }
```