**第一步：充值（如同进赌场要换筹码）：**

cleos push action dice deposit '[ "wang", "100.0000 EOS" ]' -p wang

cleos push action dice deposit '[ "zhao", "100.0000 EOS" ]' -p zhao

wang和zhao是我事先创建好的，各自都有1000个EOS。

tips: 充值之前需要取得dice智能合约的授权。

现在我们来看看deposit函数:

void deposit( const account\_name from, const asset& quantity ) {

eosio\_assert( quantity.is\_valid(), "invalid quantity" );

eosio\_assert( quantity.amount > 0, "must deposit positive quantity" );

auto itr = accounts.find(from);

if( itr == accounts.end() ) {

// 如果在dice合约里面没有账户，就创建一个

itr = accounts.emplace(\_self, [&](auto& acnt){

acnt.owner = from;

});

}

// 向dice转账，由于使用的是eos token，转账通过eosio.token合约执行

action(

permission\_level{ from, N(active) },

N(eosio.token), N(transfer),

std::make\_tuple(from, \_self, quantity, std::string(""))

).send();

// 修改dice合约中该账户的余额

accounts.modify( itr, 0, [&]( auto& acnt ) {

acnt.eos\_balance += quantity;

});

}

第二步：下注

$ openssl rand 32 -hex

271261da3bedbf0196d43dad5a73abc309e2d40750f09096c01e09b2ee0f91d5

$ echo -n '271261da3bedbf0196d43dad5a73abc309e2d40750f09096c01e09b2ee0f91d5' | xxd -r -p | sha256sum -b | awk '{print $1}'

921e0c66a8866ca0037fbb628acd5f63f3ba119962c9f5ca68d54b5a70292f36

$ cleos push action dice offerbet '[ "3.0000 EOS", "wang", "921e0c66a8866ca0037fbb628acd5f63f3ba119962c9f5ca68d54b5a70292f36" ]' -p wang

executed transaction: 77ff114b6c80bd09e453b113531eb213a555fca888451d94f703ae0dfa512b65 280 bytes 109568 cycles

# dice <= dice::offerbet {"bet":"3.0000 EOS","player":"wang","commitment":"921e0c66a8866ca0037fbb628acd5f63f3ba119962c9f5ca68..

首先是使用openssl产生一个随机数，然后求出它的哈希，最后用这个哈希下注。

现在我们来看看deposit函数，这也是dice合约最长的成员函数

void offerbet(const asset& bet, const account\_name player, const checksum256& commitment) {

eosio\_assert( bet.symbol == S(4,EOS) , "only EOS token allowed" );

eosio\_assert( bet.is\_valid(), "invalid bet" );

eosio\_assert( bet.amount > 0, "must bet positive quantity" );

eosio\_assert( !has\_offer( commitment ), "offer with this commitment already exist" );

require\_auth( player );

auto cur\_player\_itr = accounts.find( player );

eosio\_assert(cur\_player\_itr != accounts.end(), "unknown account");

// Store new offer

auto new\_offer\_itr = offers.emplace(\_self, [&](auto& offer){

offer.id = offers.available\_primary\_key();

offer.bet = bet;

offer.owner = player;

offer.commitment = commitment;

offer.gameid = 0;

});

// Try to find a matching bet

auto idx = offers.template get\_index<N(bet)>();

auto matched\_offer\_itr = idx.lower\_bound( (uint64\_t)new\_offer\_itr->bet.amount );

if( matched\_offer\_itr == idx.end()

|| matched\_offer\_itr->bet != new\_offer\_itr->bet

|| matched\_offer\_itr->owner == new\_offer\_itr->owner ) {

// No matching bet found, update player's account

accounts.modify( cur\_player\_itr, 0, [&](auto& acnt) {

eosio\_assert( acnt.eos\_balance >= bet, "insufficient balance" );

acnt.eos\_balance -= bet;

acnt.open\_offers++;

});

} else {

// Create global game counter if not exists

auto gdice\_itr = global\_dices.begin();

if( gdice\_itr == global\_dices.end() ) {

gdice\_itr = global\_dices.emplace(\_self, [&](auto& gdice){

gdice.nextgameid=0;

});

}

// Increment global game counter

global\_dices.modify(gdice\_itr, 0, [&](auto& gdice){

gdice.nextgameid++;

});

// Create a new game

auto game\_itr = games.emplace(\_self, [&](auto& new\_game){

new\_game.id = gdice\_itr->nextgameid;

new\_game.bet = new\_offer\_itr->bet;

new\_game.deadline = 0;

new\_game.player1.commitment = matched\_offer\_itr->commitment;

memset(&new\_game.player1.reveal, 0, sizeof(checksum256));

new\_game.player2.commitment = new\_offer\_itr->commitment;

memset(&new\_game.player2.reveal, 0, sizeof(checksum256));

});

// Update player's offers

idx.modify(matched\_offer\_itr, 0, [&](auto& offer){

offer.bet.amount = 0;

offer.gameid = game\_itr->id;

});

offers.modify(new\_offer\_itr, 0, [&](auto& offer){

offer.bet.amount = 0;

offer.gameid = game\_itr->id;

});

// Update player's accounts

accounts.modify( accounts.find( matched\_offer\_itr->owner ), 0, [&](auto& acnt) {

acnt.open\_offers--;

acnt.open\_games++;

});

accounts.modify( cur\_player\_itr, 0, [&](auto& acnt) {

eosio\_assert( acnt.eos\_balance >= bet, "insufficient balance" );

acnt.eos\_balance -= bet;

acnt.open\_games++;

});

}

}

下注的时候，如果找不到匹配的订单，就会保存起来；如果找到匹配订单，游戏就开始了。

第三步： 开盅

cleos push action dice reveal '[ "921e0c66a8866ca0037fbb628acd5f63f3ba119962c9f5ca68d54b5a70292f36", "271261da3bedbf0196d43dad5a73abc309e2d40750f09096c01e09b2ee0f91d5" ]' -p wang

这里需要把第二步生成的随机数和它的哈希传进去。

现在我们来看看reveal函数:

void reveal( const checksum256& commitment, const checksum256& source ) {

assert\_sha256( (char \*)&source, sizeof(source), (const checksum256 \*)&commitment );

auto idx = offers.template get\_index<N(commitment)>();

auto curr\_revealer\_offer = idx.find( offer::get\_commitment(commitment) );

eosio\_assert(curr\_revealer\_offer != idx.end(), "offer not found");

eosio\_assert(curr\_revealer\_offer->gameid > 0, "unable to reveal");

auto game\_itr = games.find( curr\_revealer\_offer->gameid );

player curr\_reveal = game\_itr->player1;

player prev\_reveal = game\_itr->player2;

if( !is\_equal(curr\_reveal.commitment, commitment) ) {

std::swap(curr\_reveal, prev\_reveal);

}

eosio\_assert( is\_zero(curr\_reveal.reveal) == true, "player already revealed");

// 正常情况下需要两个人开盅，如果一方开盅后，另一方迟迟不开，则算开盅的那一方赢

if( !is\_zero(prev\_reveal.reveal) ) {

// 最后一个玩家揭开，将2个玩家的source和commitment看作一个整体，求出它的哈希

checksum256 result;

sha256( (char \*)&game\_itr->player1, sizeof(player)\*2, &result);

auto prev\_revealer\_offer = idx.find( offer::get\_commitment(prev\_reveal.commitment) );

// 通过比较哈希的第0个字节和第1个字节的大小，决定胜负。

// 不同的数据，哈希是不一样的，这段数据由玩家1和玩家2提交的数据构成，因此玩家1和玩家2都能影响游戏的结果

int winner = result.hash[1] < result.hash[0] ? 0 : 1;

if( winner ) {

pay\_and\_clean(\*game\_itr, \*curr\_revealer\_offer, \*prev\_revealer\_offer);

} else {

pay\_and\_clean(\*game\_itr, \*prev\_revealer\_offer, \*curr\_revealer\_offer);

}

} else {

// 第一个玩家开盅，记下source，并且启动5分钟倒计时，如果第二个玩家在5分钟内没有开盅，第一个玩家赢得游戏。

games.modify(game\_itr, 0, [&](auto& game){

if( is\_equal(curr\_reveal.commitment, game.player1.commitment) )

game.player1.reveal = source;

else

game.player2.reveal = source;

game.deadline = now() + FIVE\_MINUTES;

});

}

}

上面的source和commitment对应的是随机数和它的哈希。

以上三步是dice合约的主要功能，还有许多技术细节需要自己摸索。在重写dice智能合约的过程中，我学到了很多C++ 11的编程技巧。

比如DAWN 3.0 使用eosio::multi\_index作为容器，这大大方便了开发。

如：

struct account {

account( account\_name o = account\_name() ):owner(o){}

account\_name owner;

asset eos\_balance;

uint32\_t open\_offers = 0;

uint32\_t open\_games = 0;

bool is\_empty()const { return !( eos\_balance.amount | open\_offers | open\_games ); }

uint64\_t primary\_key()const { return owner; }

EOSLIB\_SERIALIZE( account, (owner)(eos\_balance)(open\_offers)(open\_games) )

};

typedef eosio::multi\_index< N(account), account> account\_index;

account\_index accounts;

有了eosio::multi\_index，我们可以使用emplace来插入数据，使用modify来修改数据，使用erase删除数据。

还有更加精妙的例子：

struct offer {

uint64\_t id;

account\_name owner;

asset bet;

checksum256 commitment;

uint64\_t gameid = 0;

uint64\_t primary\_key()const { return id; }

uint64\_t by\_bet()const { return (uint64\_t)bet.amount; }

key256 by\_commitment()const { return get\_commitment(commitment); }

static key256 get\_commitment(const checksum256& commitment) {

const uint64\_t \*p64 = reinterpret\_cast<const uint64\_t \*>(&commitment);

return key256::make\_from\_word\_sequence<uint64\_t>(p64[0], p64[1], p64[2], p64[3]);

}

EOSLIB\_SERIALIZE( offer, (id)(owner)(bet)(commitment)(gameid) )

};

typedef eosio::multi\_index< N(offer), offer,

indexed\_by< N(bet), const\_mem\_fun<offer, uint64\_t, &offer::by\_bet > >,

indexed\_by< N(commitment), const\_mem\_fun<offer, key256, &offer::by\_commitment> >

> offer\_index;

offer\_index offers;

offer\_index类型既可以通过bet来索引，也可以通过commitment来索引。

例如在上面的offerbet函数中，使用bet进行索引

auto idx = offers.template get\_index<N(bet)>();

auto matched\_offer\_itr = idx.lower\_bound( (uint64\_t)new\_offer\_itr->bet.amount );

1

2

在上面的reveal函数中，则用commitment来索引

auto idx = offers.template get\_index<N(commitment)>();

auto curr\_revealer\_offer = idx.find( offer::get\_commitment(commitment) );