链滴

Bytom 储蓄分红 DAPP 开发指南

作者：bytom

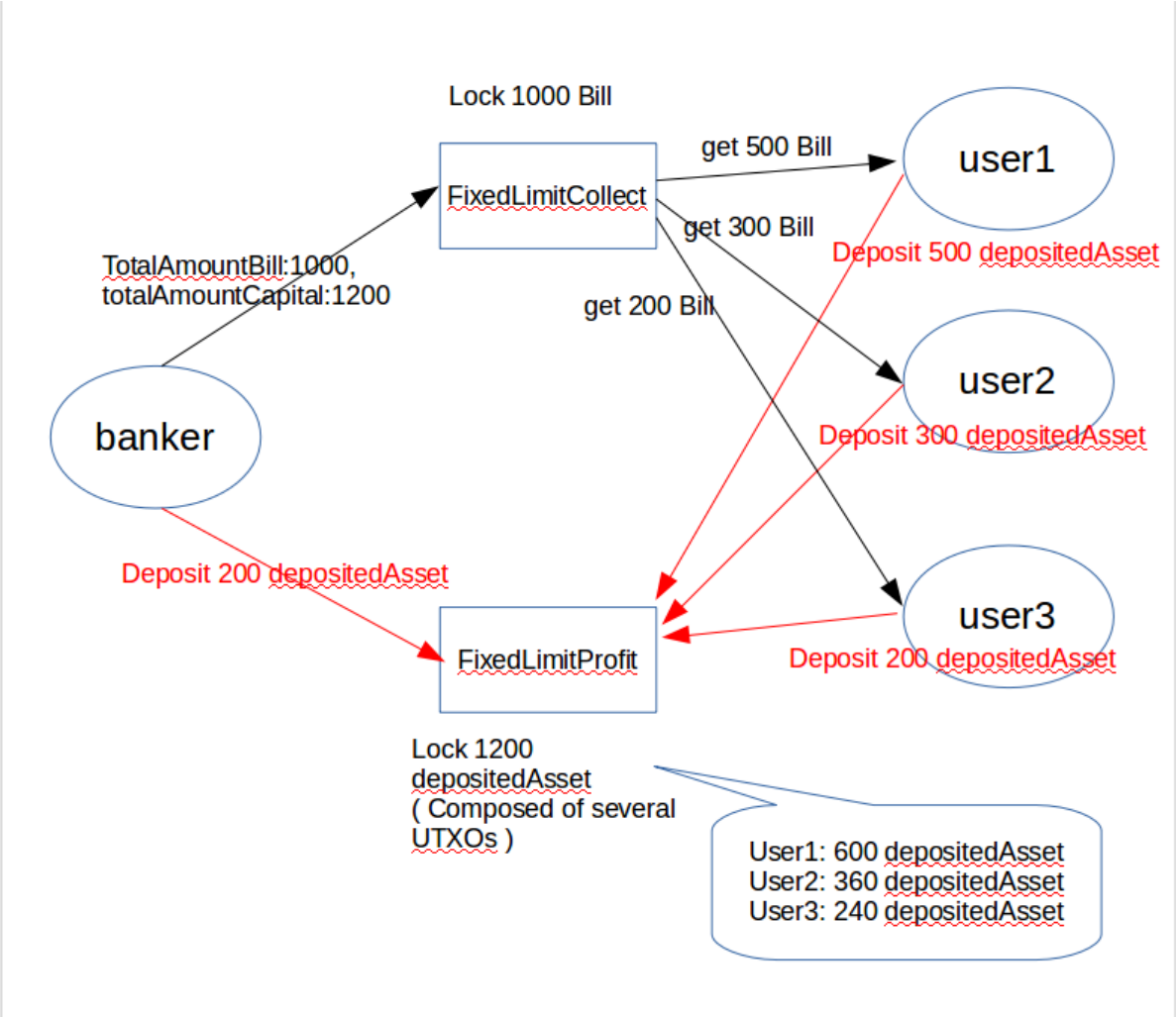
原文链接：https://ld246.com/article/1563417191367

来源网站：链滴

储蓄分红DAPP

储蓄分红合约简介

储蓄分红合约指的是项目方发起了一个锁仓计划（即储蓄合约和取现合约），用户可以在准备期自由择锁仓金额参与该计划，等到锁仓到期之后还可以自动获取锁仓的利润。用户可以在准备期内（dueBlockHeight）参与储蓄，按照合约规定可以 1：1 获取同等数量的储蓄票据资产，同时用户锁仓的资产（eposit）将放到取现合约中，并且项目方是无法动用的，等到锁仓期限（expireBlockHeight）一到用户便可以调用取现合约将自己储蓄的资产连本待息一同取出来。其示意图如下:



从上图中可以看出，项目方发布了一个利润为20%的锁仓项目，其中储蓄合约FixedLimitCollect锁定了000个票据资产（bill），同时项目方将200个储蓄资产（deposit）锁定到利息合约中。待项目方发完合约之后，所有用户便可以参与了。例如上图中user1调用合约储蓄了500，这500个储蓄资产将被定在取现合约FixedLimitProfit中，同时user1获得了500个票据资产，剩余找零的资产将继续锁定在蓄合约FixedLimitCollect中，以此类推，user2和user3也是相同的流程，直到储蓄合约没有资产为止取现合约FixedLimitProfit跟储蓄合约的模型大致相同，只是取现合约是由多个UTXO组成的，用户在现的时候可以并行操作。但是如果合约中的面值不能支持用户一次性取现的话，需要分多次提取。例如ser1拥有500个票据资产，而可以获得的本息总额为600，但是取现的UTXO面值为500,那么user1一最多只能取500,剩下的100需要再构造一笔交易来提现。

合约源代码

原文链接：Bytom 储蓄分红 DAPP 开发指南

```
// 储蓄合约
import "./FixedLimitProfit"
contract FixedLimitCollect(assetDeposited: Asset,
                           totalAmountBill: Amount,
                           totalAmountCapital: Amount,
                           dueBlockHeight: Integer,
                           expireBlockHeight: Integer,
                           additionalBlockHeight: Integer,
                           banker: Program,
                           bankerKey: PublicKey) locks billAmount of billAsset {
    clause collect(amountDeposited: Amount, saver: Program) {
        verify below(dueBlockHeight)
        verify amountDeposited <= billAmount && totalAmountBill <= totalAmountCapital
        define sAmountDeposited: Integer = amountDeposited/100000000
        define sTotalAmountBill: Integer = totalAmountBill/100000000
        verify sAmountDeposited > 0 && sTotalAmountBill > 0
        if amountDeposited < billAmount {
            lock amountDeposited of assetDeposited with FixedLimitProfit(billAsset, totalAmountB
ll, totalAmountCapital, expireBlockHeight, additionalBlockHeight, banker, bankerKey)
            lock amountDeposited of billAsset with saver
            lock billAmount-amountDeposited of billAsset with FixedLimitCollect(assetDeposited,
otalAmountBill, totalAmountCapital, dueBlockHeight, expireBlockHeight, additionalBlockHeig
t, banker, bankerKey)
        } else {
            lock amountDeposited of assetDeposited with FixedLimitProfit(billAsset, totalAmountB
ll, totalAmountCapital, expireBlockHeight, additionalBlockHeight, banker, bankerKey)
            lock billAmount of billAsset with saver
        }
    }
    clause cancel(bankerSig: Signature) {
        verify above(dueBlockHeight)
        verify checkTxSig(bankerKey, bankerSig)
        unlock billAmount of billAsset
    }
}

// 取现合约(本金加利息)
contract FixedLimitProfit(assetBill: Asset,
                           totalAmountBill: Amount,
                           totalAmountCapital: Amount,
                           expireBlockHeight: Integer,
                           additionalBlockHeight: Integer,
                           banker: Program,
                           bankerKey: PublicKey) locks capitalAmount of capitalAsset {
    clause profit(amountBill: Amount, saver: Program) {
        verify above(expireBlockHeight)
        define sAmountBill: Integer = amountBill/100000000
        define sTotalAmountBill: Integer = totalAmountBill/100000000
        verify sAmountBill > 0 && sTotalAmountBill > 0 && amountBill < totalAmountBill
        define gain: Integer = totalAmountCapital*sAmountBill/sTotalAmountBill
        verify gain > 0 && gain <= capitalAmount
        if gain < capitalAmount {
            lock amountBill of assetBill with banker
            lock gain of capitalAsset with saver
```

```
            lock capitalAmount - gain of capitalAsset with FixedLimitProfit(assetBill, totalAmountBi
l, totalAmountCapital, expireBlockHeight, additionalBlockHeight, banker, bankerKey)
        } else {
            lock amountBill of assetBill with banker
            lock capitalAmount of capitalAsset with saver
        }
    }
    clause cancel(bankerSig: Signature) {
        verify above(additionalBlockHeight)
        verify checkTxSig(bankerKey, bankerSig)
        unlock capitalAmount of capitalAsset
    }
}
```

合约的源代码说明可以具体参考Equity合约介绍.

注意事项：

- 时间期限不是具体的时间，而是通过区块高度来大概估算的（平均区块时间间隔大概为 2.5分钟）
- 比原的精度是 8, 即 1BTM = 100000000 neu，正常情况下参与计算都是以neu为单位的，然而拟机的int64类型的最大值是9223372036854775807，为了避免数值太大导致计算溢出，所以对计的金额提出了金额限制（即amountBill/100000000）
- 另外 clause cancel是项目方的管理方法，如果储蓄或者取现没有满额，项目方也可以回收剩余的产

编译并实例化合约

编译Equity合约可以参考一下Equity编译器的介绍说明。假如储蓄合约FixedLimitCollect的参数如下：

```
assetDeposited          :c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da56d4aa4e4235
3ee
totalAmountBill         :10000000000
totalAmountCapital      :20000000000
dueBlockHeight          :1070
expireBlockHeight       :1090
additionalBlockHeight   :1100
banker                  :0014dedfd406c591aa221a047a260107f877da92fec5
bankerKey               :055539eb36abcaaf127c63ae20e3d049cd28d0f1fe569df84da3aedb018ca
bf
```

其中bankerKey是管理员的publicKey，可以通过比原链的接口list-pubkeys来获取，注意管理员需保存一下对应的rootXpub和Path，否则无法正确调用clause cancel。

实例化合约命令如下：

```
// 储蓄合约
./equity FixedLimitCollect --instance c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da
6d4aa4e4235a3ee 10000000000 20000000000 1070 1090 1100 0014dedfd406c591aa221a047
260107f877da92fec5 055539eb36abcaaf127c63ae20e3d049cd28d0f1fe569df84da3aedb018c
1bf

// 取现合约
```

./equity FixedLimitProfit --instance c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da5d4aa4e4235a3ee 10000000000 20000000000 1090 1100 0014dedfd406c591aa221a047a26017f877da92fec5 055539eb36abcaaf127c63ae20e3d049cd28d0f1fe569df84da3aedb018ca1bf

发布合约交易

发布合约交易即将资产锁定到合约中。由于目前无法在比原的dashboard上构造合约交易，所以需要助外部工具来发送合约交易，比如postman。按照上述示意图所示，项目方需要发布1000个储蓄资的储蓄合约和200个利息资产取现合约。假设项目方需要发布1000个储蓄资产（假如精度为8,那么1000在比原链中表示为100000000000）的锁仓合约，那么他需要将对应数量的票据锁定在储蓄合约中，交易模板如下：


```
{
  "base_transaction": null,
  "actions": [
    {
      "account_id": "0ILGLSTC00A02",
      "amount": 20000000,
      "asset_id": "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff",
      "type": "spend_account"
    },
    {
      "account_id": "0ILGLSTC00A02",
      "amount": 100000000000,
      "asset_id": "13016eff73ffb7539a69e122f80f5c1cc94446773ac3f64dec290429f87e73b3",
      "type": "spend_account"
    },
    {
      "amount": 100000000000,
      "asset_id": "13016eff73ffb7539a69e122f80f5c1cc94446773ac3f64dec290429f87e73b3",
      "control_program": "20055539eb36abcaaf127c63ae20e3d049cd28d0f1fe569df84da3aedb
18ca1bf160014dedfd406c591aa221a047a260107f877da92fec5024c04024204022e040500c81
a8040500e40b540220c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da56d4aa4e4235
3ee4d3b02597a642f0200005479cda069c35b797ca153795579a19a695a790400e1f505965379
400e1f505967c00a07c00a09a69c35b797c9f9161644d010000005b79c2547951005e79895d79
95c79895b798959798958798957989537a894caa587a649e0000005479cd9f6959790400e1f5059653
90400e1f505967800a07800a09a5c7956799f9a6955797b957c96c37800a052797ba19a69c3787
9f91616487000000005b795479515b79c1695178c2515d79c16952c3527994c251005d79895c7
895b79895a79895979895879895779895679890274787e008901c07ec1696399000000005b79
479515b79c16951c3c2515d79c16963aa000000557acd9f69577a577aae7cac890274787e00890
c07ec169515b79c2515d79c16952c35c7994c251005d79895c79895b79895a798959798958798
57798956798955798902747 87e008901c07ec169632a020000005b79c2547951005e79895d79
95c79895b798959798958798957989537a894caa587a649e0000005479cd9f6959790400e1f5059653
90400e1f505967800a07800a09a5c7956799f9a6955797b957c96c37800a052797ba19a69c3787
9f91616487000000005b795479515b79c1695178c2515d79c16952c3527994c251005d79895c7
895b79895a79895979895879895779895679890274787e008901c07ec1696399000000005b79
479515b79c16951c3c2515d79c16963aa000000557acd9f69577a577aae7cac890274787e00890
c07ec16951c3c2515d79c169633b020000547acd9f69587a587aae7cac747800c0",
      "type": "control_program"
    }
  ],
  "ttl": 0,
  "time_range": 1521625823
}
```

合约交易成功后，合约control_program对应的UTXO将会被所有用户查询到，使用比原链的接口list-nspent-outputs即可查询。

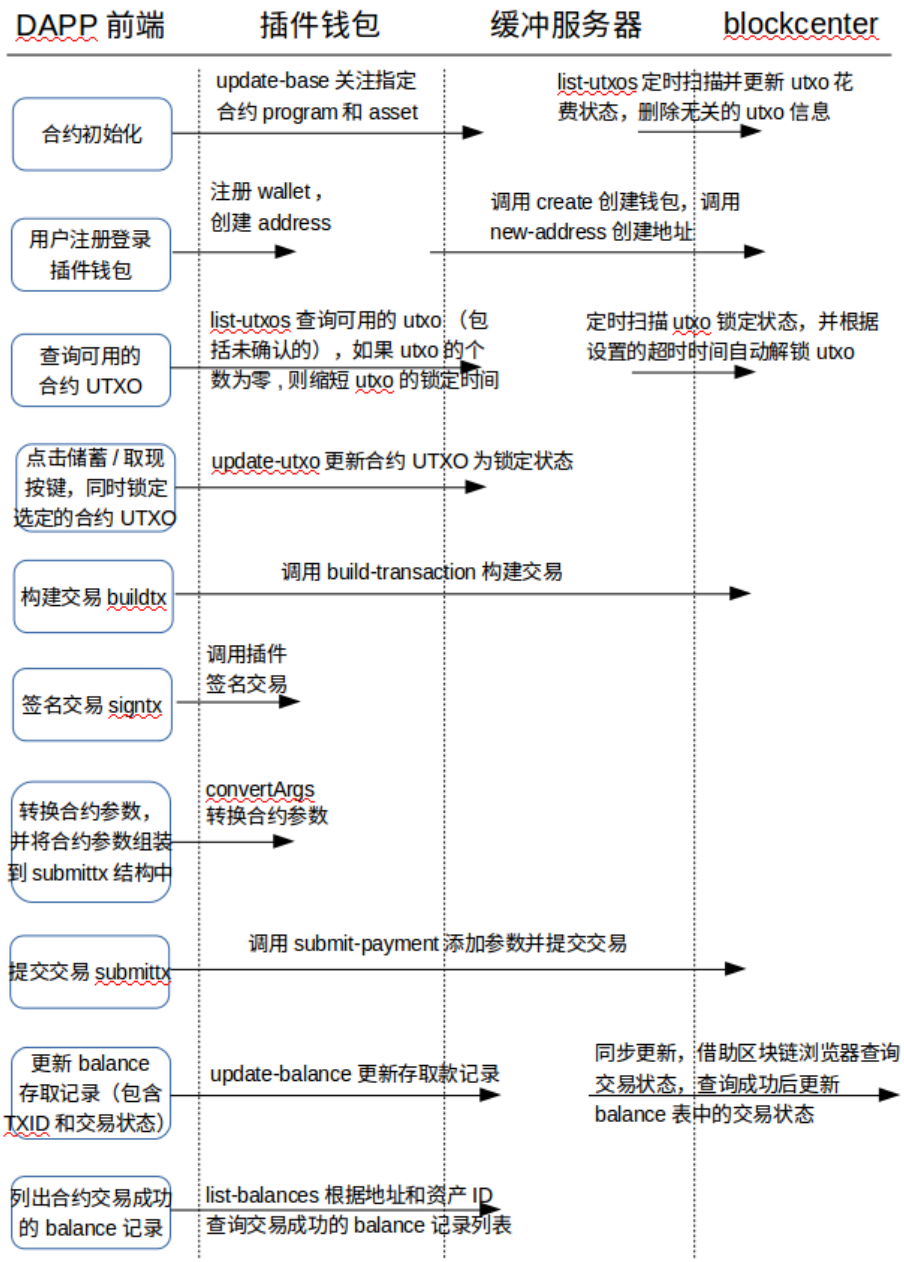
此外，开发者需要存储一下合约UTXO的assetID和program，以便在DAPP的前端页面的config配置件和bufferserver缓冲服务器中调用。如上所示：

```
// 储蓄合约
assetID：13016eff73ffb7539a69e122f80f5c1cc94446773ac3f64dec290429f87e73b3
program：20055539eb36abcaaf127c63ae20e3d049cd28d0f1fe569df84da3aedb018ca1bf160
14dedfd406c591aa221a047a260107f877da92fec5024c04024204022e040500c817a8040500e4
b540220c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da56d4aa4e4235a3ee4d3b025
7a642f0200005479cda069c35b797ca153795579a19a695a790400e1f5059653790400e1f50596
c00a07c00a09a69c35b797c9f9161644d010000005b79c2547951005e79895d79895c79895b79
9597989587989537a894caa587a649e0000005479cd9f6959790400e1f5059653790400e1f5059
7800a07800a09a5c7956799f9a6955797b957c96c37800a052797ba19a69c3787c9f9161648700
000005b795479515b79c1695178c2515d79c16952c3527994c251005d79895c79895b79895a7
8959798958798957798956798902747 87e008901c07ec1696399000000005b795479515b79c1
951c3c2515d79c16963aa000000557acd9f69577a577aae7cac890274787e008901c07ec169515
79c2515d79c16952c35c7994c251005d79895c79895b79895a7989597989587989577989567985
5579890274787e008901c07ec169632a020000005b79c2547951005e79895d79895c79895b79
9597989587989537a894caa587a649e0000005479cd9f6959790400e1f5059653790400e1f5059
7800a07800a09a5c7956799f9a6955797b957c96c37800a052797ba19a69c3787c9f9161648700
000005b795479515b79c1695178c2515d79c16952c3527994c251005d79895c79895b79895a7
8959798958798957798956798902747 87e008901c07ec1696399000000005b795479515b79c1
951c3c2515d79c16963aa000000557acd9f69577a577aae7cac890274787e008901c07ec16951c
c2515d79c169633b020000547acd9f69587a587aae7cac747800c0

// 取现合约
assetID：c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da56d4aa4e4235a3ee
program：20055539eb36abcaaf127c63ae20e3d049cd28d0f1fe569df84da3aedb018ca1bf160
14dedfd406c591aa221a047a260107f877da92fec5024c040242040500c817a8040500e40b5402
0c6b12af8326df37b8d77c77bfa2547e083cbacde15cc48da56d4aa4e4235a3ee4caa587a649e0
00005479cd9f6959790400e1f5059653790400e1f505967800a07800a09a5c7956799f9a695579
b957c96c37800a052797ba19a69c3787c9f91616487000000005b795479515b79c1695178c251
d79c16952c3527994c251005d79895c79895b79895a7989597989587989577989567989027747
7e008901c07ec1696399000000005b795479515b79c16951c3c2515d79c16963aa000000557ac
9f69577a577aae7cac747800c0
```

储蓄分红DAPP架构模型

比原链的DAPP总体框架模型描述了DAPP的大致结构模型，结合储蓄分红合约案例，其具体流程如：



DAPP前端

储蓄分红合约前端逻辑处理流程大致如下：

- 1）调用插件

比原的chrome插件源码位于Bytom-JS-SDK，开发比原DAPP时调用插件的说明可以参考Dapp Developer Guide

- 2）配置合约参数

该Dapp demo中需要配置实例化的参数为assetDeposited、totalAmountBill、totalAmountCapital、ueBlockHeight、expireBlockHeight、additionalBlockHeight、banker、bankerKey。其前端配文件为configure.json.js

```
var config = {
    "solonet": {
```

```
        "depositProgram": "2091194ddbf3614cafbadb1274c33e61afd4d5044c6ec4c30f8202980
99c30083160014c800033d5e94de5f22e23a6d3cbeaed87b55bd640600204aa9d101050010a5
4e8203310d9951697418af3cdbe7a9cdde1dc49bb5439503dacb33828d6c9ef5af5a24dfc01567
64f5010000c358797ca153795579a19a6957790400e1f5059653790400e1f505967c00a07c00a0
a69c358797c9f916164290100000005879c2547951005b79895a7989597989587989537a894c9a
67a6493000000057790400e1f5059653790400e1f505967800a07800a09a5a7956799f9a695579
b957c96c37800a052797ba19a69c3787c9f9161647c0000000059795479515979c1695178c251
b79c16952c3527994c251005b79895a7989597989587989577989567989027478 7e008901c07e
169638e0000000059795479515979c16951c3c2515b79c169639a000000567a567aae7cac8902
4787e008901c07ec169515879c2515a79c16952c3597994c251005a7989597989587989577989
67989557989027478 7e008901c07ec16963f0010000005879c2547951005b79895a7989597989
87989537a894c9a567a6493000000057790400e1f5059653790400e1f505967800a07800a09a5a
956799f9a6955797b957c96c37800a052797ba19a69c3787c9f9161647c000000059795479515
79c1695178c2515b79c16952c3527994c251005b79895a7989597989587989577989567989027
787e008901c07ec169638e0000000059795479515979c16951c3c2515b79c169639a000000567
567aae7cac890274787e008901c07ec16951c3c2515a79c16963fc010000567a567aae7cac7478
0c0",
        "profitProgram": "2091194ddbf3614cafbadb1274c33e61afd4d5044c6ec4c30f820298019
c30083160014c800033d5e94de5f22e23a6d3cbeaed87b55bd640600204aa9d101050010a5d4
820666f298d34806a6db0528c3b3d081bc00fa58aa393e7c42f90d67eb7db2a524f4c9a567a649
00000057790400e1f5059653790400e1f505967800a07800a09a5a7956799f9a6955797b957c96
37800a052797ba19a69c3787c9f9161647c0000000059795479515979c1695178c2515b79c169
2c3527994c251005b79895a7989597989587989577989567989027478 7e008901c07ec169638e
0000000059795479515979c16951c3c2515b79c169639a000000567a567aae7cac747800c0",
        "assetDeposited": "3310d9951697418af3cdbe7a9cdde1dc49bb5439503dacb33828d6c9e
5af5a2",
        "assetBill": "666f298d34806a6db0528c3b3d081bc00fa58aa393e7c42f90d67eb7db2a524f"

        "totalAmountBill": 1000000000000,
        "totalAmountCapital": 2000000000000,
        "dueBlockHeight": 0,
        "expireBlockHeight": 0,
        "banker": "0014c800033d5e94de5f22e23a6d3cbeaed87b55bd64",
        "gas": 0.4
    },
    "testnet":{
        "depositProgram": "20f39af759065598406ca988f0dd79af9175dd7adcbe019317a2d6055
8b1597ac1600147211ec12410ce8bd0d71cab0a29be3ea61c71eb103c8260203da240203da24
2060080f420e6b50600407a10f35a2000d38a1c946e8cba1a69493240f281cd925002a43b81f51
c4391b5fb2ffdacd4d4302597a64370200005479cda069c35b790400e1f5059600a05c797ba19a
3795579a19a695a790400e1f5059653790400e1f505967800a07800a09a6955797b957c9600a0
9c35b797c9f9161645b010000005b79c2547951005e79895d79895c79895b798959798958798
537a894ca4587a64980000005479cd9f6959790400e1f5059653790400e1f505967800a07800a0
a5c7956799f9a6955797b957c967600a069c3787c9f91616481000000005b795479515b79c169
178c2515d79c16952c3527994c251005d79895c79895b79895a7989597989587989577989567
890274787e008901c07ec1696393000000005b795479515b79c16951c3c2515d79c16963a4000
00557acd9f69577a577aae7cac890274787e008901c07ec169515b79c2515d79c16952c35c7994
251005d79895c79895b79895a7989597989587989577989567989557989027478 7e008901c07
c1696332020000005b79c2547951005e79895d79895c79895b798959798958798 7537a894ca4
87a64980000005479cd9f6959790400e1f5059653790400e1f505967800a07800a09a5c7956799
9a6955797b957c967600a069c3787c9f91616481000000005b795479515b79c1695178c2515d7
c16952c3527994c251005d79895c79895b79895a7989597989587989577989567989027478 7e
08901c07ec1696393000000005b795479515b79c16951c3c2515d79c16963a4000000557acd9f
9577a577aae7cac890274787e008901c07ec16951c3c2515d79c1696343020000547acd9f69587
```

```
587aae7cac747800c0",
        "profitProgram": "20f39af759065598406ca988f0dd79af9175dd7adcbe019317a2d605578
1597ac1600147211ec12410ce8bd0d71cab0a29be3ea61c71eb103c8260203da2402060080f42
e6b50600407a10f35a20f855baf98778a892bad0371f5afca845191824dc8584585d566fbbc8ef1
304c4ca4587a64980000005479cd9f6959790400e1f5059653790400e1f505967800a07800a09a
c7956799f9a6955797b957c967600a069c3787c9f91616481000000005b795479515b79c16951
8c2515d79c16952c3527994c251005d79895c79895b79895a79895979895879895779895679 8
0274787e008901c07ec1696393000000005b795479515b79c16951c3c2515d79c16963a400000
557acd9f69577a577aae7cac747800c0",
        "assetDeposited": "00d38a1c946e8cba1a69493240f281cd925002a43b81f516c4391b5fb2f
dacd",
        "assetBill": "f855baf98778a892bad0371f5afca845191824dc8584585d566fbbc8ef1f304c",
        "totalAmountBill": 100000000000000,
        "totalAmountCapital": 200000000000000,
        "dueBlockHeight": 140506,
        "expireBlockHeight": 140506,
        "banker": "00147211ec12410ce8bd0d71cab0a29be3ea61c71eb1",
        "gas": 0.4
    }
}
```


● 3）前端预计算处理

以储蓄合约FixedLimitCollect为例，前端需要对该合约进行verify语句的预判断逻辑，以防用户输入数之后执行失败。此外，合约中billAmount of billAsset表示锁定的资产和数量，而billAmount、billsset和utxohash都是储存在缓冲服务器的数据表里面，因此前端需要调用list-utxo查找与该资产asset和rogram相关的所有未花费的utxo。 具体可以参考DAPP DEMO前端案例。

● 4）交易组成

比原的交易是多输入多输出的模板结构，如果合约中包含了多个lock或unlock语句，那么就需要用户造多输入多输出的交易模板，同时，构造交易还需要根据lock语句或unlock语句来变换。交易构造具可以参考储蓄合约交易模型和取现合约交易模型的前端源代码。

- 交易　input结构如下：

spendUTXOAction(utxohash)表示花费的合约utxo，其中utxohash表示合约UTXO的hash，而spenWalletAction(amount, Constant.assetDeposited)表示用户输入的储蓄或取现的数量（仅包含中需资产交换的合约中），而资产类型则由前端固定。

```
export function spendUTXOAction(utxohash){
    return {
        "type": "spend_utxo",
        "output_id": utxohash
    }
}

export function spendWalletAction(amount, asset){
    return {
        "amount": amount,
        "asset": asset,
        "type": "spend_wallet"
    }
}

const input = []
input.push(spendUTXOAction(utxohash))
```

```
input.push(spendWalletAction(amount, Constant.assetDeposited))
```

- 交易　output结构如下：

根据合约中if-else判定逻辑，下面便是储蓄分红合约的output的构造模型。

```
export function controlProgramAction(amount, asset, program){
    return {
        "amount": amount,
        "asset": asset,
        "control_program": program,
        "type": "control_program"
    }
}

export function controlAddressAction(amount, asset, address){
    return {
        "amount": amount,
        "asset": asset,
        "address": address,
        "type": "control_address"
    }
}

const output = []
if(amountDeposited < billAmount){
    output.push(controlProgramAction(amountDeposited, Constant.assetDeposited, Constant.
rofitProgram))
    output.push(controlAddressAction(amountDeposited, billAsset, saver))
    output.push(controlProgramAction((billAmount-amountDeposited), billAsset, Constant.de
ositProgram))
}else{
    output.push(controlProgramAction(amountDeposited, Constant.assetDeposited, Constant.
rofitProgram))
    output.push(controlAddressAction(billAmount, billAsset, saver))
}
```

- 5）启动前端服务

编译前端命令如下：

```
npm run build
```

启动之前需要先启动bufferserver缓冲服务器，然后再启动前端服务，其前端启动命令如下：

```
npm start
```

DAPP缓冲服务器

缓冲服务器主要是为了在管理合约UTXO层面做一些效率方面的处理，包括了对bycoin服务器是如何步请求的，此外对DAPP的相关交易记录也进行了存储。具体可以参考一下bufferserver源代码。

- 1）储蓄分红合约的架构说明如下：

- 缓冲服务器构成，目前设计了　3张数据表：base、utxo和balance表。其中base表用于初始化该APP关注的合约program，即在查找utxo集合的时候，仅仅只需过滤出对应的program和资产即可; u

xo表是该DAPP合约的utxo集合，其数据是从bycoin服务器中实时同步过来的，主要是为了提高DAPP并发性; balance表是为了记录用户参与该合约的交易列表。

● 后端服务由 API进程和同步进程组成，其中API服务进程用于管理对外的用户请求，而同步进包含了两个方面：一个是从bycoin服务器同步utxo，另一个是则是通过区块链浏览器查询交易状态

● 项目管理员调用 update-base接口更新DAPP关注的合约program和asset。而utxo同步进程根据base表的记录来定时扫描并更新本地的utxo表中的信息，并且根据超时时间定期解锁被锁定的uto

● 用户在调用储蓄或取现之前需要查询合约的 utxo是否可用，可用的utxo集合中包含了未确认的txo。用户在前端在点击储蓄或取现按键的时候，会调用utxo最优匹配算法选择最佳的utxo，然后调用pdate-utxo接口对该utxo进行锁定，最后就用户就可以通过插件钱包调用bycoin服务器的构建交易口来创建交易、签名交易和提交交易。倘若所有合约utxo都被锁定了，则会缩短第一个utxo的锁定时为60s，设置该时间间隔是为了保证未确认的交易被成功验证并生成未确认的utxo。如果该时间间隔没有产生新的utxo，则认为前面一个用户并没有产生交易，则60s后可以再次花费该utxo。

● 用户发送交易成功后会生成两条 balance记录表，默认状态是失败的，其中交易ID用于向区块浏览器查询交易状态，如果交易成功则会更新balance的交易状态。此外，前端页面的balance列表只显示交易成功的记录。

● 2）编译 bufferserver源代码

按照README安装部署服务需要的软件包Mysql和Redis，然后下载源代码并编译：

```
make all
```

编译完成之后，在target目录下会生成可执行文件api和updater。

● 3）启动服务

使用root用户创建数据库和数据表，其命令如下：

```
mysql -u root -p < database/dump.sql
```

修改配置文件config_local.json，字段说明参考README的config配置参数详解。

启动api和updater服务器，其中api是提供JSON RPC请求的服务进程，updater是提供同步blockcener和区块链浏览器数据请求的服务进程。

```
./target/api config_local.json
```

```
./target/updater config_local.json
```