三公技术文档

V1.0

修订历史

版本&日期	修订内容
v1.0：2019-4-12	初稿
v1.0：2019-4-22	修订
v1.0：2019-4-23	改取牌规则，由之前的hash映射改为映射和区间取牌
v1.0：2019-4-24	改发牌规则，用一副牌来发牌，牌有花色
v1.0：2019-4-25	改取牌规则，根据变动的reveal随机数来取牌

1. 引言

1.1. 目的

本文为主要描述三公合约的需求、合约接口以及实现方案。

本文档的阅读对象包括项目设计、开发与测试人员、平台维护与运营人员，以及第三方开发者。

1.2. 定义、简写和缩略语

- Excellencies:Excellencies

1.3. 参考资料

- 网络资料

2. 需求说明

2.1. 需求概述

2.1.1. 游戏流程：

- 选择筹码-->下注-->确认支付。三公游戏由智能合约控制，实现公平公开的开牌机制，玩家可以选择一个或多个闲家进行投注，如果玩家投注的闲家比庄家的牌大则获胜，赔率1:1。游戏每一分钟一期。

2.1.2. 游戏机制：

- **玩家**：每一期有一个庄家和多个玩家，每一期允许下注多次，到期后由庄家调用合约触发开奖与结算。
- **发牌**：轮流发牌，每次发一张牌，庄家和闲家总牌总数为三张。
- **牌型**：牌张J、Q、K为公牌，公牌不计点数，10不算点数也不是公牌，牌A--9对应1--9点，把三张牌的点数相加，
- **大小规则**： 大三公>小三公>混三公>点数大小

  大三公：KKK>QQQ>JJJ

  小三公：101010>999>888>777>666>555>444>333>222>AAA

  混三公：KKQ>KKJ>KQQ>KQJ>KJJ>QQJ>QJJ

  点 数：9点>8点>7点>6点>5点>4点>3点>2点>1点>0点.点数相同，则比较公牌数量，2张>1张>0张，如公牌张数一样，则比较第一张公牌大小，如第一张公牌相同，则比较最大公牌的花色,花色的大小顺序是(黑红梅方)

- **结算**：游戏下注完成后，根据庄家和玩家的牌面到期后由庄家开奖结算，或者由玩家调用结算接口结算庄家未结算的赌注。

2.1.3. 奖池与费用:

- 每期游戏结束，平台均会扣除赢家奖金的2%，作为奖池。

  (1) 最近的单次最高奖金玩家（倒计时的24小时内没有其他玩家获得相同或更多的奖金）,则获得50%奖金池作为奖励,出现新的最高奖金玩家则倒计时重置为24小时。 举例说明：从0时开始算起，23时是最高投注为100dc且获胜,23.40时另外投注玩家投注110dc，且获胜，那么此时，起始时间置位0，重新开始计算，直到24小时内没有玩家超过他的下注，24小时到点后，将奖池的一半派奖给110dc的投注玩家。这个24小时返奖接口，是游戏平台记录数据并调用的的，合约只负责校验和执行转币操作。

  (2)牌型奖励：每一局比赛玩家出现三公（大三公/小三公/混三公)，则可获得5%奖池奖励，同一局比赛出现多位玩家三公，则根据投注大小共同瓜分5%奖池奖励。

- 每次下注时，游戏平台均会扣除5%，以帮助我们支付账单并使游戏可以运行。

2.1.4. 公平公开的开牌机制:

- 整个游戏玩法都由智能合约进行控制，以BCB主网产生的“哈希值”进行发牌，从闲家一张一张发，每家三张。哈希值是由BCB全球节点打包区块时轮流产生，它是根据一系列加密算法生成的不可控随机结果，保证三公游戏的公平公开，每一局的输赢结果，玩家都可自主验证，一切数据也将不可篡改的链上记录为准。
- 洗牌，在游戏的开奖的时候，会把上一局的牌(52张),重新打乱，根据区块链的随机数生成一副新的牌。
- 哈希值有64个字符，根据变换参数reveal，求出随机数，根据随机数模上剩余牌的张数，以此做为下标取牌，这种模式直至取完15张牌，然后在按闲家先发牌的规则，每个人依次发完牌。

假如发牌结果为：

台位/类型	闲一	闲二	闲三	闲四	庄家
牌张	8,Q,5	7,Q,7	K,2,4	10,Q,J	k,Q,10
点数	3 点	4 点	6 点	0 点	0 点
结果	赢	赢	赢	输	输三家赢一家

- 开奖结果举例说明(点数相同的情况)

台位/类型	闲一	庄家
牌张	8,Q,5	7,K,6
点数	3 点	3 点
结果	输	赢
庄赢 点数都是三 但是 庄的公牌是K大于闲家的Q		

台位/类型	闲一	庄家
牌张	8,Q,5	J,K,3
点数	3 点	3 点
结果	输	赢
庄赢 庄家的公牌数多		

台位/类型	闲一	庄家
牌张	8,Q(红桃),5	5,Q(黑桃),8
点数	3 点	3 点
结果	输	赢
庄赢 庄家的Q黑桃大于红桃		

- 投注结算举例说明：闲家投注“B”100dc，获胜后100x2为200，减去投注的手续费100x5/100为195，减去要给奖池的金额195x2/100，转给

玩家的金额为 191.1，如果闲家的牌为大小三公或混三公，闲家将和所有投注“B”的玩家按投注金额多少共同瓜分百分之5的奖池金额。

2.2. 异常场景

- 用户下注时账户余额不足时，下注失败。
- 用户下注时投入的代币智能合约不支持时，下注失败。
- 用户下注交易的区块高度大于庄家承诺的区块高度时，下注失败。
- 用户下注交易提供的庄家承诺数据不正确时，下注失败。
- 用户下注时智能合约账户中未锁定资金不足以支付用户可能赢取的奖金时，下注失败。
- 如果用户下注失败，用户的账户余额不会发生变化。
- 用户下注成功后，如果庄家没有及时结算，超时以后，用户可以构造一笔退款交易广播到BCBChain上调用智能合约进行退款。

3. 合约设计

3.1 合约元数据

- 合约名称：**Excellencies**
- 合约版本：1.0
- 组织：orgNUjCm1i8RcoW2kVTbDw4vKW6jzfMxewJHjkhuiduhjuikjuyhnnjkuhujk111【组织名称：example】
- 状态数据库KEY前缀：/orgNUjCm1i8RcoW2kVTbDw4vKW6jzfMxewJHjkhuiduhjuikjuyhnnjkuhujk111/SicBo

3.2. 合约存储数据

3.2.1. secretSigner

- 验签公钥数据
- **键**: <前缀>/secretSigner
- **值**:

```
//@:public:store:cache
secretSigner types.PubKey               // 存储验签公钥
```

3.2.2. lockedInBets

- 被锁定的资金总额
- **键**: <前缀>/lockedInBets/<代币名称>
- **值**:

```
//@:public:store:cache
lockedInBets map[string]bn.Number //存储代币对应的被锁定的资金总额 key=代币名称
```

3.2.3. settings

- 合约配置信息
- **键**: <前缀>/settings
- **值**:

```
type Settings struct {
    LimitMaps   map[string]Limit  `json:"limitMaps"`   // Maximum winning amount、Maximum bet、Minimum bet limit (cong)
    FeeRatio    int64   `json:"feeRatio"`// 中奖后手续费比例（千分比）
    FeeMiniNum  int64   `json:"feeMiniNum"`  // 发送给clt最小的数量 (cong)
    SendToCltRatio   int64  `json:"sendToCltRatio"`  // 手续费中发送给clt的部分（千分比）
    BetExpirationBlocks  int64  `json:"betExpirationBlocks"` // 开奖的区块间隔
    TokenNames  []string  `json:"tokenNames"`  //支持的代币名称列表
}

type Limit struct {
```

```
    MaxProfit   int64   `json:"maxProfit"`   // 最大中奖金额 （cong）
    MaxLimit    int64   `json:"maxLimit"`// 最大下注限额(cong)
    MinLimit    int64   `json:"minLimit"`// 最小下注限额单位（cong）
}

//@:public:store:cache
settings *Settings  // 存储合约配置参数
```

3.2.4. recFeeInfo

- 接收手续费分成的信息
- **键:** <前缀>/recFeeInfo
- **值:**

```
type RecFeeInfo struct {
    RecFeeRatio int64   `json:"recFeeRatio"` // 手续费分配比例表(‰)
    RecFeeAddr  types.Address   `json:"recFeeAddr"`  // 接收手续费的地址列表
}

//@:public:store:cache
recFeeInfo []RecFeeInfo // 存储手续费分成信息,所有分成比例之和小于等于100%
```

3.2.5. roundInfo

- 存储轮信息
- **键:** <前缀>/roundInfo/<当前轮commit随机数>
- **值:**

```
type RoundInfo struct {
    Commit   []byte   `json:"commit"`   // 当前游戏随机数hash值
    TotalBuyAmount   map[string]bn.Number   `json:"totalBuyAmount"`   // 当前的总投注金额 map key：tokenName(币种名称)
    TotalBetCount  int64   `json:"totalBetCount"`// 当前总投注数量
    State   int64   `json:"state"`// 当前轮状态 0 未开奖 1已开奖 2 已退款 3开奖中
    ProcessCount  int64   `json:"processCount"` // 当前状态处理投注数量（结算、退款记下标）
    FirstBlockHeight  int64   `json:"firstBlockHeight"` // 当前轮初始化时的区块高度，判断是否超时使用
    Settings  Settings   `json:"settings"` // 当前轮的配置信息
    BetInfos  map[string]map[string]BetInfo   `json:"betInfos"` // 赌注信息 Key1 player address, key2 currency name
    Game  Game   `json:"game"` // Game Result, Maker and Player Type
    PoolFlag  bool   `json:"poolFlag"`  // 是否会有大小三公 混三公结算标志 默认是false  结算完置位true
    BetAmountByModel  map[string]map[string]bn.Number    `json:"betAmountByModel"`   //不同牌型 币种投注总额
    OriginPokers  []*Poker   `json:"originPokers"` //原始牌面，没经过排序的牌

}

//赌注信息
type BetInfo struct {
    TokenName  string   `json:"tokenName"` //玩家投注币种名称
    Gambler   types.Address   `json:"gambler"`   // 玩家投注地址
    Amount    bn.Number    `json:"amount"`// 玩家投注总金额
    BetData   []BetData   `json:"betData"`   // 玩家投注详情
    WinAmount   bn.Number   `json:"winAmount"` // 玩家本次投注最大奖金
    Settled   bool  `json:"settled"`   // 当前投注是否已经结算
}

//下注信息结构体
type BetData struct {
    BetMode   string    `json:"betMode"`   // A代表庄家 B, C, D, E代表玩家
    BetAmount bn.Number `json:"betAmount"` // 投注金额
}

//@:public:store
    roundInfo map[string]*RoundInfo //key1轮标识
```

3.3. 合约方法设计

- **初始化函数**

方法名称	描述
InitChain	合约初始化操作，合约部署(升级)时自动执行一次。

- **方法列表**

方法名称	描述	gas	owner	游戏后台	普通用户
SetSecretSigner	设置验签公钥	500	√	×	×
SetOwner	设置合约拥有者	500	√	×	×
SetSettings	设置合约运行参数	500	√	×	×
SetRecFeeInfo	设置手续费分成比例	500	√	×	×
PlaceBet	用户下注	500	×	√	√
SettleBet	开奖并结算	500	×	√	×
RefundBets	超时退款	500	×	√	√
WithdrawFunds	提取游戏收益	500	√	×	×
WithdrawWin	玩家开奖	500	×	√	√
SlipperHighestTransfer	24小时返奖	500	×	√	×
CarveUpPool	大小三公 混三公派奖	500	×	√	×

3.3.1. 典型场景

- **暂无图片**

3.3.2. 游戏投注信息

- **无**

3.3.3. InitChain

方法原型

```
    //@:constructor
    func (sg *Excellencies) InitChain()
```

功能说明

- 初始化游戏配置参数。

前置任务

- 部署合约

输入参数

- 无

处理流程

- 构造存储数据对象`settings`并赋初始值，然后保存。

```
settings := SBSettings{}
settings.FeeRatio = 50
settings.FeeMiniNum = 300000
settings.SendToCltRatio = 100
settings.BetExpirationBlocks = 250
settings.TokenNames = []string{sb.sdk.Helper().GenesisHelper().Token().Name()}

limitMaps :=sb.createLimitMaps(settings.TokenNames)
settings.LimitMaps = limitMaps
sb._setSettings(&settings)
```

约束条件

- 无

输出结果

- 无

后置任务

- 无

异常处理

- 无

3.3.4. SetSecretSigner

方法原型

```
    //@:public:method:gas[500]
    func (sg *Excellencies) SetSecretSigner(newSecretSigner types.PubKey)
```

功能说明

- 设置用于验证随机数签名的公钥。

前置任务

- 无

输入参数

- newSecretSigner 指定的公钥

处理流程

- 判断输入参数是否合法有效。
- 更新存储数据对象`secretSigner`，然后保存。

约束条件

- 只有合约拥有者可调用该方法。
- 输入的公钥长度必须满足32字节。

输出结果

- 自定义收据`setSecretSigner`，格式如下：

```
// Name of Receipt: setSecretSigner
type emitSetSecretSigner struct{
    NewSecretSigner types.PubKey  `json:"newSecretSigner"`      // 新的验签公钥
}
```

后置任务

- PlaceBet

异常处理

- 不满足约束条件直接引发panic

3.3.5. SetSettings

方法原型

```
    //@:public:method:gas[500]
    func (sb *SicBo) SetSettings(newSettingsStr string)
```

功能说明

- 设置游戏的运行参数。

前置任务

- 无

输入参数

- newSettingsStr 指定的游戏的运行参数json格式字符串，示例如下：

```
{
"limitMaps": {
"LOC": {
    "maxProfit": 2000000000000,
    "maxLimit": 20000000000,
    "minLimit": 100000000
}
},
    "feeRatio": 50,
    "feeMiniNum": 300000,
    "sendToCltRatio": 100,
    "betExpirationBlocks": 250,
    "tokenNames": ["LOC"]
}
```

处理流程

- 判断输入参数是否合法有效。
- 判断当前状态是否允许修改游戏参数。
- 更新存储数据对象`settings`，然后保存。

约束条件

- 只有合约拥有者可调用该方法。
- 当游戏支持的所有代币的锁定金额都为0时才能修改游戏参数。
- 输入参数必须符合json格式。
- 输入参数必须满足如下条件：
  - 支持的代币名称列表长度必须大于0且名称必须有效；
  - 最大盈利大于0；
  - 单注最大限额>0；
  - 单注最小限额>0并小于最大限额；
  - 最小手续费限额>0；
  - 手续费比例0-1000（‰）；
  - 转给clt的手续费比例0-1000（‰）；
  - 开奖超时的间隔区块>0。
  - 代币地址必须大于0

输出结果

- 自定义收据setSettings，结构如下：

```
// Name of Receipt: setSettings
type setSettings struct {
        TokenNames  []string  `json:"tokenNames"`
        Limit   map[string]Limit  `json:"limit"`
        FeeRatio  int64    `json:"feeRatio"`
        FeeMiniNum  int64   `json:"feeMiniNum"`
        SendToCltRatio  int64   `json:"sendToCltRatio"`
        BetExpirationBlocks int64   `json:"betExpirationBlocks"`
    }
```

后置任务

- 无

异常处理

- 不满足约束条件直接引发panic

3.3.6. SetRecFeeInfo

方法原型

```
    //@:public:method:gas[500]
    func (sg *Excellencies) SetRecFeeInfo(recFeeInfoStr string)
```

功能说明

- 设置手续费的分成比例及接收分成的账户地址。

前置任务

- 无

输入参数

- recvFeeInfosStr 分成配置json串,示例：

```
  [
  {
    "ratio":500,
    "address":"bcbKrHJUVGAt4R9gcfsBthu3dWJR7bAYq1c8",
    },
    {
    "ratio":500,
    "address":"bcbNwdwjpDotDDLGiB9pARk1CcSM71bdgTef",
    },
  ]
```

处理流程

- 判断输入参数是否合法有效。
- 更新存储数据对象`setRecFeeInfo`，然后保存。

约束条件

- 只有合约拥有者可调用该方法。
- 输入参数必须符合json格式。
- 设置参数必须保证地址符合指定格式。
- 至少有一条分成信息。
- 接收分成的账户地址不能是本合约的账户地址。
- 各分成比例相加后小于等于1000.

输出结果

- 自定义收据setRecFeeInfo，结构如下：

```
type RecFeeInfo struct {
    RecFeeRatio int64   `json:"recFeeRatio"` // 手续费分配比例表(‰)
    RecFeeAddr  types.Address   `json:"recFeeAddr"`  // 接收手续费的地址列表
}

// Name of Receipt: setRecFeeInfo
type setRecFeeInfo struct {
    Info []RecFeeInfo   `json:"info"`
}
```

后置任务

- 无

异常处理

- 不满足约束条件直接引发panic。

3.3.7. WithdrawFunds

方法原型

```
//@:public:method:gas[500]
func (sg *Excellencies) WithdrawFunds(tokenName string, beneficiary types.Address, withdrawAmount bn.Number)
```

功能说明

- 合约拥有者从合约账户中提取游戏的收益。

前置任务

- 无

输入参数

- tokenName 指定代币名称
- beneficiary 提取收益到指定的账户地址
- withdrawAmount 提取金额（单位cong）

处理流程

- 判断输入参数是否合法有效。
- 从合约账户向指定账户转账。

约束条件

- 只有合约拥有者可调用该方法。
- 代币名称必须是标准代币名称。
- 提取收益的地址必须是个有效账户地址。
- 提取金额需要满足如下条件：
  - 提取金额必须大于0；
  - 不能大于合约账户余额扣除锁定金额后的值。

输出结果

- 从合约账户到指定账户的转账收据std::transfer。
- 自定义收据withdrawFunds，结构如下：

```
  // Name of Receipt: withdrawFunds
type withdrawFunds struct {
  TokenName   string    `json:"tokenName"`
  Beneficiary  types.Address    `json:"beneficiary"`
  WithdrawAmount   bn.Number    `json:"withdrawAmount"`
}
```

后置任务

- 无

异常处理

- 不满足约束条件时引发panic。

3.3.8. PlaceBet

方法原型

```
//@:public:method:gas[500]
func (sg *Excellencies) PlaceBet(betJson string,
commitLastBlock int64, commit, signData []byte, refAddress types.Address)
```

功能说明

- 用于提交用户下注信息。

前置任务

- SetSecretSigner 必须设置验签公钥

输入参数 - betJson 为押注的json序列化字符串

```
  [
{
  "betMode":"A",
  "betAmount":100000000
},
{
 "betMode":"B",
 "betAmount":100000000
}
  ]
```

- commitLastBlock 投注是的区块高度
- commit 随机数hash
- signData 随机数签名
- refAddress 推荐人地址

处理流程

-校验 -语法校验 判断验签公钥是否设置并校验投注在区块链上确认的最大有效高度与随机数hash的签名是否合法。 -逻辑校验 是否存在当前轮 校验传递的参数是否合法 检测庄家是否有足够的金钱赔偿 检测是否区块过期 -拿到投注信息校验 tokenName amount 校验 -算出可能中奖的金额 ===》中奖金额不能超过最大maxProfit（获利） -数据存储 保存投注信息 -发送收据

约束条件

- 投注账户拥有足够的资金用于支付投注费用。
- 合约所有者不能参与投注。
- 投注金额 模式 必须合法有效。
- 推荐人的地址合法有效

输出结果

- 自定义收据placeBet，结构如下：

```
// Name of Receipt: placeBet
type placeBet struct {
    TokenName   string   `json:"tokenName"`
    Gambler  types.Address  `json:"gambler"`
    TotalMaybeWinAmount   bn.Number   `json:"totalMaybeWinAmount"`
    BetDataList   []BetData   `json:"betDataList"`
    CommitLastBlock   int64   `json:"commitLastBlock"`
    Commit   []byte   `json:"commit"`
    SignData  []byte   `json:"signData"`
    RefAddress   types.Address   `json:"refAddress"`
}
```

后置任务

- 无

异常处理

- 不满足约束条件时引发panic。

3.3.9. SettleBet

```
//@:public:method:gas[500]
func (sg *Excellencies) SettleBet(reveal []byte, settleCount int64)
```

功能说明

- 当前投注进行开奖和结算，并清除对应的投注信息。

前置任务

- PlaceBet

输入参数 -reveal 随机数 -settleCount 第几回合

处理流程

-判断是否下注成功 下注里有没有数据 -判断有没有过期
-根据随机数开奖 获取开奖信息 使用随机数reveal生成对应的hash（变量名为commit），并使用hash结果取得指定轮的信息。 - 根据投注信息判断投注状态： - 指定投注已开奖并结算完成的情况下则返回错误信息 - 指定投注未开奖并超时则返回错误信息 - 未开奖则进行开奖结算方法如下： - 首先根据开奖号码计算用户的中奖金额。 - 根据配置计算奖池金额费用、是否获得奖池金额。 - 根据用户下注金额计算总的手续费，再根据单注金额计算需要从中奖奖金中扣除的部分。 - 根据配置计算clt分红和分成数额。 - 根据计算结果进行转账。 - 将状态设置为已经开完奖的状态 - 保存当前投注更新后信息并产生开奖或结算收据。

约束条件 - 只有庄家才能调用此方法

输出结果

- 自定义收据settleBet，结构如下：

```
// Name of Receipt: settleBet
    type settleBet struct {
        TokenName  []string   `json:"tokenName"`
        Commit  []byte   `json:"commit"`
        Game   Game   `json:"game"`
        WinNumber  int64   `json:"winNumber"`
        TotalWinAmount  map[string]bn.Number   `json:"totalWinAmount"`
        Finished   bool   `json:"finished"`
        PoolAmount   map[string]bn.Number   `json:"poolAmount"`
        OriginPokers   []*Poker   `json:"OriginPokers"`

    }

//庄家闲家的牌信息集合
type Game struct {
    Gamer  map[string]*PokerSet
    Pp    *PokerPool
}
//玩家的牌面信息
type PokerSet struct {
    No string
    Pokers []*Poker
    IsWin  bool
    TypeSan  uint8  //THREESAME MIXTHREE THREENOSAME  代表三张牌一样  混三公  点数
    ThreeSum uint8 // three porkers in sum
}
//需要发牌的全部牌信息
type PokerPool struct {
    Pool []*Poker
}

//单个扑克信息
type Poker struct {
    Name string
    Flag uint8
    Type  uint8  //牌的颜色 3 2 1 0 分别表示黑 红 梅 方
}
```

后置任务

- 无

异常处理

- 不满足约束条件时引发panic。

3.4.0. SlipperHighestTransfer

```
    //@:public:method:gas[500]
    func (sg *Excellencies) SlipperHighestTransfer(tokenName string, playersAddress types.Address)
```

功能说明

- 24小时最高投注，奖池返奖逻辑

前置任务

-

输入参数 -tokenName 币种 -playersAddress 地址

处理流程

-地址校验 -参数校验 -时间校验 **约束条件** - 只有庄家才能调用此方法

输出结果

- 自定义收据slipperHighestTransfer，结构如下：

```
// Name of Receipt: slipperHighestTransfer
type slipperHighestTransfer struct {
    TokenName  string  `json:"tokenName"`
    PlayersAddress types.Address   `json:"playersAddress"`
    WinningAmount  bn.Number   `json:"winningAmount"`
}
```

后置任务

- 无

异常处理

- 不满足约束条件时引发panic。

3.4.1. WithdrawWin

```
    //@:public:method:gas[500]
    func (sg *Excellencies) WithdrawWin (commit []byte)
```

功能说明

- 庄家看到牌面对自己不利，不想开奖，玩家此时可以自己开奖

前置任务

-

输入参数 -commit 此轮游戏的提交

处理流程
-是否存在该轮 -区块高度校验 -该轮状态校验

约束条件 - 玩家可以调用

输出结果

- 参考开奖输出，和庄家开奖一样

后置任务

- 无

异常处理

- 不满足约束条件时引发panic。

3.4.2. CarveUpPool

```
    //@:public:method:gas[500]
    func (sg *Excellencies) CarveUpPool (commit []byte)
```

功能说明

- 瓜分奖池

前置任务

-

输入参数 -commit 轮信息的查询参数

处理流程 -校验 -获取应该瓜分的奖池金额 -算出那些投资模型具备分奖资格 -根据具有派奖资格的模型，算出这些投注里各个币种的投注总额 -算出每个币种每一份投注应该分多少奖金，（当奖池金额很小 *1000 这样避免了得出的结果是nil） -遍历投注信息，判断时候有派奖资格，根据投注金额，算出应该派多少奖金（要除以1000），然后转账

约束条件 -地址校验 -状态校验 -是否具有派奖条件验证

输出结果

```
    //setOwner receipt
     type carveUpPool struct {
        CountNum    int64  `json:"countNum"`  //有多少笔转账
        AmountPoolList  map[string]bn.Number   `json:"amountPoolList"`  //分了多少奖金
    }
```

后置任务

- 无

异常处理

- 不满足约束条件时引发panic。