# 政务链（GAChain）技术文档

- If(condition, iftrue, iffalse)
- Mult(num1,num2)
- Trim(text)

- Value mapping | 值映射
  - Address([wallet_id])
  - Money(value)
  - Date(date,[format])
  - DateTime(datetime,[format])
  - Now([format])
- HTML元素
  - A(class,text, href)
  - Div(class,text)
  - Divs(class,[class,]) ... DivsEnd
  - P(class, text)
  - Em(class,text)
  - Small(class,text)
  - Span(class,text)
  - Strong(class,text)
  - Label(text,[class])
  - Legend(class, text)
  - Tag(tagname, [text], [class])
  - Image(src, [alt], [class] )
  - MarkDown(text)
  - Val(idname)
- 条件结构
  - If(condition) ... Else ... ElseIf ... IfEnd
- 显示表单元素
- Form(class) ... FormEnd
  - Input(idname,[class],[placeholder],[type],[value])
  - Textarea(idname,[class],[value])
  - InputAddress(idname,[class],[value] )
  - InputDate(idname,[class],[value] )
  - InputMoney(idname,[class],[value])
  - Select(idname, list, [class], [value])
  - TextHidden(idname,....)
  - Source(idname,[value])
- 从数据库获取值
  - ValueById(table,idval,columns,[aliases])

- GetList(name, table, colnames, [where], [order], [limit])
- ListVal(name, index, column])
- ForList(name) ... FormListEnd
- GetOne(colname, table, where, [value])
- GetRow(prefix, table, colname, [value])
- StateVal(name, [index])
- Table

- 显示合约
  - BtnContract(contract, name, message, params, [class], [onsuccess], [pageparams])
  - TxButton
  - TxForm
- 导航元素
  - Navigation(params, ...)
  - LinkPage(page,text,[params])
  - LiTemplate(page, [text], [params], [class])
  - BtnPage(page, name,[params],[class], [anchor])
  - BtnEdit( page, icon, [params] )
    - Back(page, [params])
- 格式化页面模板
  - - PageTitle(header) ... PageEnd()
    - Title(text)
  - FullScreen(state)
  - WhiteMobileBg(state)
- 组织多级菜单
  - MenuItem(title, page, [params], [icon])
  - MenuGroup(title,[idname],[icon]) ... MenuEnd:
  - MenuBack(title, [page])
  - MenuPage(page)
- 数据表示
  - Ring(count,[title],[size])
  - WiAccount(address)
  - WiBalance(value, money)
  - WiCitizen(name, address, [image], [flag])
  - Map(coords)
  - MapPoint(coords)
  - ChartPie
  - ChartBar

# 政务链（GAChain）项目描述

## 关于政务链（GAChain）

## 合约语言

合约是政务链（GAChain）算法实现的基本结构，完整代码片段确保了可接受从其他用户或其他合约中输入的数据，根据其正确和交易的情况做出正确分析，以此作为起草合约的基础。合约语言是一种脚本语言，可快速编译字节编码，支持主要类型值的变量、涉及功能、一组变准运算符和结构，以及一个bug管理的数据库。

合约编译成字节代码，可供所有用户使用。 在合约显示时，创建具有epy输入数据和一组变量的隔离堆栈，执行字节码时由虚拟机处理，因此，多进程不会相互影响，甚至在处理同一字节代码时，也不会相互影响。

#### 数值类型和变量

语言变量由数值类型标识来宣示，自动类型的转换是典型案例的应用，数值的使用类型如下：

- bool - boolean，取值为true或false；
- bytes - 字节序列；
- int - 64位整数；

- address - 64位无符号整数;
- array - 具有任意类型数值值的数组;
- map - 具有字符串键、任意类型数值的关联数组;
- money - 大整数类型的整数；这些数值存储在数据库中，且没有小数点，它们根据货币的设置在界面中显示时插入;
- float - 64位浮点数；* 字符串 - 单行；以双引号或反引号指定 - “This is a line”或This is a line。

所有标识符：变量、函数、合约等，其名称都需要区分大小写（如 MyFunc 和 myFunc，此为不同的标识符）。

变量使用var关键字表示，后跟变量名、名称及其类型，变量被定义后，在大括号内操作。 在描述变量时，默认值会自动分配：对于bool类型为fales，对于所有数字类型（零值和字符串，以及空字符串）都是false。 变量声明的例子：

```
func myfunc( val int) int {
    var mystr1 mystr2 string, mypar int
    var checked bool
    ...
    if checked {
         var temp int
         ...
    }
}
```

**数组**

该语言支持两种类型的数组：

- array - 具有数字索引的简单数组，以0开头;
- map - 具有字符串键的关联数组。

通过在方括号中指定索引来分配和接收组件。

```
var myarr array
var mymap map
var s string

myarr[0] = 100
myarr[1] = "This is a line"
mymap["value"] = 777
mymap["param"] = "Parameter"

s = Sprintf("%v, %v, %v", myarr[0] + mymap["value"], myarr[1], mymap["param"])
// s = 877, This is a line, Parameter
```

### “If” 和 “while” 结构

合约描述语言包含条件结构有if和while循环结构，它们在函数中和合约中使用。 这些结构可以彼此插入。

关键字必须后跟条件表达式样，如果条件表达式返回一个数字，则在0值被认为是false。 例如，val == 0 等同于 !val，而 val != 0 与简单的val相同。 if结构可能有一个else块，如果条件 if 语句为 fales 则执行。 比较操作可以在条件表达式中使用： <、 >、 >=、 <=、 ==、 !=，与 || (OR) 和 && (AND) 相同。

```
if val > 10 || id != $citizen {
  ...
} else {
  ...
}
```

while 结构旨在用于循环实现，只要条件为 true，则执行 while 区块。 break语句用于阻止区块内的循环。 continue语句首先用于实现循环区块。

```
while true {
    if i > 100 {
       break
    }
    ...
    if i == 50 {
       continue
    }
    ...
}
```

除了条件表达式，该语言还支持标准算术运算：+、 -、*、/。

### 函数

该函数通过使用func关键字，后跟函数名称，括号中以逗号分隔，传输的参数指示类型，以及右括号后的返回值类型定义。 功能体用大括号括起来。 如果函数没有参数，可以省略括号。 return 关键字用于从函数返回值。

任何功能性的误差，将自动触发合同停止执行，并显示一条消息。

```
func myfunc(left int, right int) int {
    return left*right + left - right
```

```
}
func test int {
    return myfunc(10, 30) + myfunc(20, 50)
}
func ooops {
    error "Ooops..."
}
```

### 合约

合约是基本的语言结构，有助于在界面中实现用户或其他合约发起的单一操作。 整个应用程序源代码作为一个有效的合约系统，通过数据库进行交互，或通过在合约主体中显示彼此。

合约由合约关键字定义，后跟合约的名称。 合约机构用大括号括起来。 合约由三部分组成：

1. 数据用于描述输入数据（变量名和类型）；
2. 条件实现输入数据的验证和确认；
3. 行为包含对合约行为的描述。

合约结构：

```
contract MyContract {
    data {
        FromId address
        ToId   address
        Amount money
    }
    func conditions {
        ...
    }
    func action {
    }
}
```

#### 数据部分的数据描述

在数据部分中描述合约输入的数据，以及用于接收数据形式的参数。 数据一行列出：如首先指定了变量名称（只有变量，但不传输数组），则可以通过双引号的间距，用于指定接口形式的构建类型和参数：

- hidden - 隐藏元素的形式；
- optional - 不用强制填写的表单；
- date - 日期和时间字段；
- polymap - 具有坐标和区域选择的地图；
- map - 具有标记地点能力的地图，如百度地图、谷歌地图；

- image - 图片上传；
- text - 在textarea字段中输入HTML代码的文本；
- address - 用于输入钱包地址的字段；
- signature:contractname - 显示合约名称的行，需要签名（在特殊描述部分进行详细说明）。

```
contract my {
  data {
      Name string
      RequestId address
      Photo bytes "image optional"
      Amount money
  }
  ...
}
```

**合约中的变量**

在数据部分中描述的输入数据合约，被发送到具有这些名称与在其前面 $ 符号变量的其它部分。

预置变量包含关于从其他显示合约中的交易，预置变量也可在合约中使用。

- $time - 内部交易时间
- $state - 内部国家标识
- $province 内部省份标识
- $block - 区块编号，含 int. 标识的转账是封闭的内部转账
- $citizen - 签署了内部转账的公民地址
- $wallet - 由钱包签署了交易的地址，如果合约是跨国的，则 state == 0
- $wallet_block - 包含交易的已确认区块的节点地址
- $block_time - 区块确认的时间，包含当前内部合约交易的区块

```
contract my {
  data {
      Name string
      Amount money
  }
  func conditions {
      if $Amount <= 0 {
         error "Amount cannot be 0"
      }
      $ownerId = 1232
  }
  func action {
      DBUpdate(Table("mytable"), $ownerId, "name,amount", $Name, $Amount - 10 )
      DBUpdate(Table("mytable2"), $citizen, "amount", 10 )
  }
}
```

**条件部分**

所获得数据的验证在本节中进行，以下命令用于警告是否存在错误，分别有：错误、警告、信息。 实际上，所有这些都会导致停止合约操作的错误，在界面中显示不同的消息：严重错误、警告和信息错误。 例如，

```
if fuel == 0 {
      error "fuel cannot be zero!"
}
if money < limit {
      warning Sprintf("该账户余额不足: %v < %v", money, limit)
}
if idexist > 0 {
      info "您已经注册"
}
```

**附加合约**

合约中的**条件**和**动作**部分可能会在另一个合约中，为此，必须指定此类合约的名称，并在括号中说明必要的参数：传输数据的名称（从显示合约的**数据**部分）用引号括起来，用 逗号 分开，后跟一个变量列表，其中包含传输的值。 例如，

```
MoneyTransfer(“SenderAccountId,RecipientAccountId,Amount”,sender_id,recipient_id,$
Price)
```

附加合约可以通过其中声明的全局变量（前面的** $ **符号）返回其中获得的值。通过通过字符串变量传送合约名称的** CallContract（）**函数，也可以显示附加的合约。

**合约签署**

由于合约书写的语言允许履行附加的合约，所以可以在不被外部用户所知，以及未授权的情况下进行交易，即其附加合约依然可以履行，让我们描述一下该种转账情形。

我们假设有一个MoneyTransferd的合约为MoneyTransfer：

```
contract MoneyTransfer {
    data {
      Recipient int
      Amount    money
    }
    ...
}
```

如果在用户发起的合约中，字符串MoneyTransfer（“Recipient,Amount”，12345,100）被刻录，则将100个币转移到钱包12345。在这种情况下，签署外部合约的用户将不会知道该笔交易。如果MoneyTransfer 合约在其中要求额外的用户签名时，可能会排除这种情况。 因此执行如下：

1. 在“MoneyTransfer”合约的数据部分添加一个名为Signature的可选参数，并且隐藏参数的字段，这样就不需要在直接调用合约时要求额外的签名，因为签名字段中的签名不涉及。

```
contract MoneyTransfer {
    data {
      Recipient int
      Amount    money
      Signature string "optional hidden"
    }
    ...
}
```

1. 添加签名表（在GAchain客户端的**签名**页面上），其条目包含：

- *MoneyTransfer* 合约名称，
- 字段名称的值将显示会给用户，其文本描述和文本将在区块确认后显示。

在当前示例中，指定了两个字段 **Receipient** 和 **Amount**：

- **Title**：您是否同意向收件人汇款?
- **Parameter**：接收文本：电子钱包ID
- **Parameter**：金额文本：金额（GAC）

现在，如果插入 *MoneyTransfer(“Recipient, Amount”, 12345, 100)* 的合约调用，将显示系统错误“*签名未定义*”。如果合约被称为如下：*MoneyTransfer(“Recipient, Amount, Signature”, 12345, 100, ”xxx...xxxxx”)*，系统错误将在签名验证后发生。在合约调用时，会验证以下信息：“初始交易的时间、用户ID、签名表中指定的字段的值”，并且签名不可能伪造。

为了让用户在 *MoneyTransfer* 合约调用时查看汇款确认，需要添加一个任意名称和类型字符串，以及可选参数 *signature:contractname* 的字段。在调用附带的 *MoneyTransfer* 合约后，只需要转发此参数。需要提醒的是，担保合约的参与参数也必须在外部合同的数据部分描述（有可能被隐藏，但仍然会在确认后显示）。例如，

```
 contract MyTest {
  data {
      Recipient int "hidden"
      Amount  money
      Signature string "signature:send_money"
  }
  func action {
      MoneyTransfer("Recipient,Amount,Signature",$Recipient,$Amount,$Signature)
  }
```

当发送一个名为 *MyTest* 的测试合约时，将向用户请求向指定钱包汇款的附加确认。 如果附带合约中列出了其他值，例如 *MoneyTransfer("Recipient,Amount,Signature",*$Recipient$*,Amount+10, $Signature)*，则会发生无效的签名错误。

**错误处理**

任何功能的参数错误都将被自动处理，可使合约停止执行，并显示相关的错误信息。

### 访问系统组件的权限

GAchain具有多级权限管理系统，用于创建和编辑数据库表、智能合约、页面和界面菜单，以及国家、省份设置模式的参数。在国家、省份设置（智能合约、表、接口）的相关部分"权限"字段中创建和编辑上述元素时，将指定权限。权限被记录为逻辑表达式，如果表达式在访问时表达式为 true ，则将其提供权限。如果"Permissions"字段保持为空，那么它将自动变为false，并且执行相关操作的访问权将完全关闭。

确定以下操作的权限：

- 列表权限 - 更改列表中的值的权限;
- 表插入权限 - 在表中写入新命令的权限;
- 新建列表权限 - 添加新列表的权限;
- 更改表权限的条件 - 更改第1-3段所列权力的权限;
- 变更智能合约的条件 - 改变合约的权力;
- 更改条件 - 更改界面页面的权力;
- 更改条件菜单 - 更改菜单的权力;
- 更改国家、省份参数的条件 - 更改国家、省份设置模式的某个参数的权限。

提供权限的最简单的方法是在"Conditions"字段中处理逻辑表达式 *$citizen == 2263109859890200332*，并指明特定用户的标识号。使用ContractAccess("NameContract") 函数，其中有权执行适当操作的合约列表被转移到其中，这是定义权限的多用途和推荐方法。例如，在金额栏的"Conditions"字段中处理ContractAccess("MoneyTransfer") 后的帐目表中，金额价值的变更将仅适用于MoneyTransfer智能合约（涉及到将资金从一个帐户转移到另一个帐户，只能通过显示的MoneyTransfer合约）。获得合约条件的在条件部分得到控制，可能相当复杂，涉及许多其他智能合约和智能律法。

为了解决系统操作的冲突或危险情况，国家和省份参数表包括特定参数（*state_changing_smart_contracts*，*state_changing_tables*，*state_changing_pages*），其中指定了获取对任何智能合约、表或页面的访问权限的条件。 这些权力是由特殊的智能律法规定的，例如规定司法裁决或负责人的一些签名。

通过使用合约来确保权力，我们获得了灵活可调的资源访问控制系统，允许自动跟踪当前从一个人转移到另一个人，例如在更换所占据的位置时。

# 嵌入式合约语言功能

合约语言的函数使用合约数据部分中获得的数据来执行操作，它们可以从数据库中读取值和写入值，也可以转换值类型，并建立合约之间的关系。

函数不返回错误 - 所有错误都会自动检查。 当在任何函数中产生错误时，合约将停止工作，并在特殊窗口中显示错误的描述。

### 从数据库中检索值

#### DBAmount(tblname string, column string, id int) money |

该函数返回money类型的数量列值，并通过表中指定列的值来搜索记录。（DBInt() 和 DBIntExt() 函数返回的 int 值不能用于获取 money int ）。

- tblname - 数据库中表的名称;
- column - 即将搜索记录的列名称;
- id - 搜索记录的值，column=所选 id 。

```
mymoney = DBAmount("dlt_wallets"), "wallet_id", $wallet)
```

#### StateVal(name string) string

该函数从国家、省份设置（state_parameters）中返回指定参数的值。

- name - 接收参数的名称。

```
Println( StateVal("gov_account"))
```

#### DBInt(tblname string, name string, id int) int

该函数通过记录的指定ID从数据库表中返回一个数值。

- tblname – 数据库中表的名称;
- name - 将取得返回值的列名称;
- id - 将被取值记录ID字段的标识符。

```
var val int
val = DBInt(Table("mytable"), "counter", 1)
```

#### DBIntExt(tblname string, name string, val (int|string), column string) int

该函数从数据库表中返回数值，并通过指定的字段和值搜索记录。

- tblname - 数据库中表的名称；
- name - 将取得返回值的列名称；
- val - 搜索记录的值。
- column - 将搜索的记录的列名称；该表必须具有此列的索引。

```
var val int
val = DBIntExt(Table("mytable"), "balance", $wallet, "wallet_id")
```

#### DBIntWhere(tblname string, name string, where string, params ...) int

该函数从数据库表的列中返回一个数值，并按照where指定的条件搜索记录。

- tblname - 数据库中表的名称；
- name - 将取得返回值的列名称；
- val - 检索记录的查询条件；字段的名称位于比较符号的左侧；字符 ？ 或 $ 被用于替换参数；
- params - 在给定的序列中，替换查询条件的参数。

```
var val int
val = DBIntWhere(Table("mytable"), "counter",  "idgroup = ? and statue=?", mygroup
, 1 )
```

#### DBString(tblname string, name string, id int) string

该函数通过从数据库表的列所记录的ID中返回一个字符串值。

- tblname - 数据库中表的名称；
- name - 将取得返回值的列名称；
- id - 从where的值中获取记录ID字段中的标识符。

```
var val string
val = DBString(Table("mytable"), "name", $citizen)
```

#### DBStringExt(tblname string, name string, val (int|string), column string) string

该函数从数据库表中返回一个字符串值，并通过指定的字段和值搜索记录。

- tblname - 数据库中表的名称；
- name - 将取得返回值的列名称；

- val - 搜索记录的值；
- column - 将搜索记录的列名称；该表必须具有此列的索引。

```
var val string
val = DBStringExt(Table("mytable"), "address", $Company, "company" )
```

#### DBFreeRequest(tblname string, val (int|string), column string)

该函数检查指定记录的存在，并且执行成本为零。旨在初步验证合约参数，以防止“垃圾”。此功能只能在合约中调用一次。如果找到具有此列值的记录，则合约将继续工作。否则，此函数将生成错误。

- tblname - 数据库中表的名称；
- name - 将取得返回值的列名称；
- val - 搜索记录的值。
- column - 将搜索的记录的列名称；该表必须具有此列的索引。

#### DBStringWhere(tblname string, name string, where string, params ...) string

该函数从数据库表的列中返回一个字符串值，并根据其中指定的条件搜索记录。

- tblname - 数据库中表的名称；
- name - 将取得返回值的列名称；
- where - 检索记录的查询条件；字段的名称位于比较符号的左侧；字符 ？ 或 $ 被用于替换参数；
- params - 在给定的序列中，替换查询条件的参数。

```
var val string
val = DBStringWhere(Table("mytable"), "address",  "idgroup = ? and company=?",
       mygroup, "My company" )
```

#### DBGetList(tblname string, column string, offset int, limit int, order string, where string, params ...) array

该函数从数据库表的列中返回一个字符串值，并根据其中指定的条件搜索记录。

- tblname - 数据库中表的名称；
- column - 将从其中获取值的列名称；
- offset - 补偿开始选择的记录；
- limit - 接收到的记录数量；如果不需要限制，则参数值为-1；
- order - 按列排序；可以是空字符串；
- where - 检索记录的查询条件；字段的名称位于比较符号的左侧；字符？或 $ 被用于替换参数；

- params - 在给定的序列中，替换查询条件的参数。

```
var ret array
ret = DBGetList(Table("mytable"), "name", 0, -1, "", "idval > ? and idval <= ? and
 company=?",
                 10, 200, "My company")
```

#### DBGetTable(tblname string, columns string, offset int, limit int, order string, where string, params ...) array

该函数返回关联映射数组，其中包含一个通过where指定条件后，获取的已被表记录的列表数值清单。关联数组中的所有值都是类型字符串，因此它们将随后被放置成适当的类型。

- tblname – 数据库中表的名称；
- columns - 以逗号分隔的接收列的名称；
- offset - 补偿开始已选择的记录；
- limit - 收到的记录数量；如果不需要限制，则参数值为 -1；
- order - 按列排序；可以是空字符串；
- where - 检索记录的查询条件；字段的名称位于比较符号的左侧；字符 ? 或 $ 被用于替换参数；
- params - 在给定的序列中，替换查询条件的参数。

```
var ret array
ret = DBGetTable(Table("mytable"), "name,idval,company", 0, -1, "", "idval > ? and
 idval <= ? and company=?",
                 10, 200, "My company")
var i int
while i<Len(ret) {
    var row map

    row = ret[i]
    myfunc(Sprintf("%s %s", row["name"], row["company"]), Int(row["idval"]) )
    i++
}
```

### 更改表中的值

#### DBInsert(tblname string, params string, val ...) int

该函数将一条记录添加到指定的表中，并返回插入记录的ID。

- tblname - 数据库中表的名称；
- params - 以逗号分隔的列名称，val 中列出的值将被写入；

- val - 参数中列出的以逗号分隔的值列表；值可以是字符串或数字。

```
DBInsert(Table("mytable"), "name,amount", "John Dow", 100)
```

#### DBUpdate(tblname string, id int, params string, val...)

该函数用于更改记录中具有指定ID的表中的列值。

- tblname - 数据库中表的名称；
- id - 可更改记录的标识符ID；
- params - 要更改的列，以逗号分隔的名称列表；
- val - 参数中列出的指定列的值列表；可以是字符串或数字。

```
DBUpdate(Table("mytable"), myid, "name,amount", "John Dow", 100)
```

#### DBUpdateExt(tblname string, column string, value (int|string), params string, val ...)

该函数用于更新具有指定值记录中的列。该表应具有指定列的索引。

- tblname - 数据库中表的名称；
- column - 将搜索记录的列名称。
- value - 用于搜索列中记录的值。
- params - 以逗号分隔的名称列表，其中 val 指定的值将被写入；
- val - 在params中列出的列记录值列表；可以是字符串或数字。

```
DBUpdateExt(Table("mytable"), "address", addr, "name,amount", "John Dow", 100)
```

### 合约回调

#### CallContract(name string, params map)

该函数通过其名称调用合约。合约部分数据中指定的所有参数应列在传输的数组中。

- name - 正在调用的合约的名称；
- params - 与合约输入数据的关联数组。

```
var par map
par["Name"] = "My Name"
CallContract("MyContract", par)
```

#### ContractAccess(name string, [name string]) bool

该函数检查执行合约的名称是否与参数中列出的名称之一相匹配。 通常用于控制合约到表的访问。 该功能在编辑表列或“权限”表中的“插入”和“新建”列字段时，在“Table permission. section”字段中指定。

- name - 合约名称。

```
ContractAccess("MyContract")
ContractAccess("MyContract","SimpleContract")
```

#### ContractConditions(name string, [name string]) bool

该函数从具有指定名称的合约中调用条件部分。对于这样的合约，数据区块必须为空。如果条件执行没有错误，则返回 true。如果在执行过程中产生错误，父合约也将以此错误结束。此功能通常用于控制表的访问权限，并且可以在编辑系统表时在“权限”字段中调用该函数。

- name – 合约名称。

```
ContractConditions("MainCondition")
```

### 带有变量值的操作

#### AddressToId(val int) int

#### Float(val int|string) float

该函数将转换整数 int，或将字符串转换为浮点数。

- val - 整数或字符串。

```
val = Float("567.989") + Float(232)
```

#### HexToBytes(hexdata string) bytes

该函数将具有十六进制编码的字符串转换为字节值（字节序列）。

- hexdata - 一个包含十六进制符号的字符串。

```
var val bytes
val = HexToBytes("34fe4501a4d80094")
```

#### Int(val string) int

该函数将字符串值转换为整数。

- val - 包含数字的字符串。

```
mystr = "-37763499007332"
val = Int(mystr)
```

#### Len(val array) int

该函数返回指定数组中的元素数。

- val - 数组。

```
if Len(mylist) == 0 {
  ...
}
```

#### PubToID(hexkey string) int

该函数通过十六进制编码的公钥返回钱包地址。

- hexkey - 十六进制形式的公钥。

```
  var wallet int
   wallet = PubToID("fa5e78.....34abd6")

AddressToId(address string) int
==============================
该函数通过公民的钱包地址字符串值，返回其身份信息号码。

Sha256(val string) string
```

- **Sha256**哈希计算数值的输入行。

```
var sha string
sha = Sha256("Test message")
```

#### Sprintf(pattern string, val ...) string

该函数基于指定的模板和参数形成一个字符串，可以使用 %d(number)、%s(string)、

%f（float）、%v(任意类型)。

- pattern - 用于形成字符串的模板。

```
out = Sprintf("%s=%d", mypar, 6448)
```

#### Str(val int|float) string

该函数将数字 int 或 float 值转换为字符串。

- val - 一个整数或一个浮点数。

```
myfloat = 5.678
val = Str(myfloat)
```

#### Table(tblname) string

该函数返回一个表的完整名称，其中带有国家号码的数字前缀，在其中调用合约，并在前缀和名称之间加上“下划线”符号。 允许使合约变得独立的国家。

- tblname - 下划线符号后，数据库中表的名称的一部分。

```
Println( Table("citizens")) // may be 1_citizens or 2_citizens etc.
```

### 更新平台元素

#### UpdateContract(name string, value string, conditions string)

该函数更新指定的合约（无法通过**DBUpdate**功能修改合约）。

- name - 合约名称;
- value - 合约文本;
- conditions - 修改合约的访问权限。

```
UpdateContract("MyContract", source, "ConditionsContract($citizen)")
```

#### UpdateMenu(name string, value string, conditions string)

该函数更新指定的菜单（无法通过**DBUpdate**功能更改菜单）。

- name - 要更新的菜单的名称;
- value - 菜单文本;
- conditions - 更改菜单的访问权限。

```
UpdateMenu("main_menu", mymenu, "ConditionsContract($citizen)")
```

#### UpdatePage(name string, value string, menu string, conditions string)

该函数更新指定的页面（该页面不能通过**DBUpdate**功能更改）。

- name - 正在更新的页面名称;
- value - 页面的文本;
- menu - 绑定到页面的菜单名称;
- conditions - 更改页面的访问权限。

```
UpdatePage("default_dashboard",mypage, "main_menu", "ConditionsContract($citizen)"
)
```

#### UpdateParam(name string, value string, conditions string)

该函数更新state_parameters表中的state_parameters（不能通过**DBUpdate**函数更改参数）。

- name - 参数名称;
- value - 参数值;
- conditions - 更改参数的访问权限。

```
UpdateParam("state_flag", $flag, "ContractConditions(`MainCondition`)")
```

## 页面模板引擎

### 通用函数描述

#### 函数类型

应用页面模板是通过一组可创建GAchain应用程序界面的专用语言的函数来创建的。 根据执行的操作类型，函数可以分为以下几组:

- 从数据库获取值;
- 处理变量的格式和值;

- 表格和图表中的数据呈现；
- 建立表格，并附有所需的一套字段，用于输入合约数据；
- 显示导航元素和显示合同；
- 创建HTML页面布局 - 具有指定CSS类的能力的不同容器；
- 实现页面模板片段的条件显示；
- 创建多级菜单。

### 函数格式

FuncName() 和 FuncName{} 两种格式用于实现页面模板构建语言的功能。 在第一种情况下，参数作为字符串数组传输；在第二种情况下作为具有命名参数（键）和值的关联数组。值不能用引号括起来。如果该值包含一个逗号或一个闭括号，它应该用双引号或后缀（``）引用。 如果函数没有参数或只有一个参数，则可以放置一个冒号而不是括号：MyFunc:param ie 相当于 MyFunc(param)。

例如，

```
FuncName( string 1, string 2, "Text, text")
```

```
FuncName{ ParamName1: string 1, ParamName2: string 2, ParamName3: "Text, text" }
```

### HTML元素的KClasses

用于创建页面布局和导航系统的HTML元素的功能包含css类的名称参数，这些参数将通过空格枚举。 目前，使用Angular Bootstrap Angle的类。 除了类之外，这些参数可以包括将被插入到相关元素中的附加属性。 属性值应使用等号标示。 如果属性没有值，则应保留等号，但不得在其后面显示（只有当属性以data-开头时，才能忽略相等的标记）。 例如，

```
Div(panel data-widget=panel-scroll myclass data-sweet-alert=, Text)
```

将会变成

```
<div class="panel myclass" data-sweet-alert data-widget="panel-scroll">Text</div>
```

### Using language resources | 使用语言资源

如果在语言表资源（政府信息中心菜单中的语言）中指定了此元素的转化，则模板引擎支持以任意语言替换页面文本元素的功能。 对于转化，文本资源的名称必须用符号 "" （ 例如Call contract$ ）或 LangRes（“Call contract”）功能进行构图。

Label()、Select()、StateValue() 等功能以及菜单创建功能会自动用语言资源替换文本。

```
* Note

Functions in the future can be supplemented with new parameters.
```

## 变量操作

#### SetVar( name=value,.....)

该函数将值分配给变量，而不在页面上显示。

- name - 变量的名称，
- value - 值；如果值包含逗号，那么它必须包含在反引号``中；如果需要替换表达式值，则使用格式为 #=，而不是 =。

例如，

```
SetVar( var1= value1, var2 = "Value 2", var3=10, `var4 #= #citizen_id#, #state_id#
` )
```

将来，变量可能被援引为 #var1#、#var2#...

#### StateLink(prefix,name)

该函数显示名称为 prefix_name 变量的含义。

## 值的操作

#### And(param, [param,...])

该函数返回逻辑运算 AND 的结果，所有列在括号中的参数都以逗号分隔。 假设参数值为 false ，如果它等于一个空字符串（“”），则为 0 或为 false 。 在所有其他情况下，假定参数值为 true 。

**Or(param, [param,...])**

该函数返回逻辑运算 OR 的结果，所有列在括号中的参数都以逗号分隔。 假设参数值为 false ， 如果它等于一个空字符串（“”），则为 0 或为 false 。 在所有其他情况下，假定参数值为 true 。

**CmpTime(time1,time2)**

该函数将相同格式的两个时间值进行比较（最好为YYYY-MM-DD HH：MM：SS格式，但也可以是任意序列格式，从年到秒， 例如，YYYYMMDD）。 它返回：

- -1 - time1 < time2,
- 0 - time1 = time2,
- 1 - time1 > time2.

**If(condition, iftrue, iffalse)**

该函数根据条件的真实性或虚假性显示两个值之一。

- condition - 条件表达式，如果等于空字符串，则为false；
- iftrue - 条件为true时返回的值；
- iffalse - 条件为false时返回的值。

允许函数附件。

**Mult(num1,num2)**

该函数显示将两个数字相乘的结果（参数可以是小数），四舍五入到最接近的整数。

**Trim(text)**

该函数将从文本行的开头和结尾移除空格及不可见字符。

## Value mapping | 值映射

**Address([wallet_id])**

该函数显示将两个数字相乘的结果（参数可以是小数），四舍五入到最接近的整数。

**Money(value)**

该函数显示货币格式的值，小数位由 state_parameters 表中的 money_digit 参数值定义。

#### Date(date,[format])

该函数以指定的格式显示日期值。

- date - 标准格式为 2006-01-02T15:04:05；
- format - 格式模板：YY短年，YYYY全年，MM - 月，DD - 日，例如 DD.MM.YY。 如果未指定格式，则将使用在语言表中指定的 dateformat 参数值，如果不存在，则为YYYY-MM-DD。

#### DateTime(datetime,[format])

该函数以指定的格式显示日期和时间值。

- datetime - 标准格式的时间为 2006-01-02T15:04:05；
- format - 格式模板：YY短年，YYYY全年，MM - 月，DD - 日，例如 DD.MM.YY。 如果未指定格式，则将使用在语言表中指定的 timeformat 参数值，如果不存在，则 YYYY-MM-DD HH:MI:SS。

#### Now([format])

该函数以指定格式显示当前时间，默认情况下以UNIX格式显示（自1970年以来的秒数），如果 datetime 指定为格式，则日期和时间显示为 YYYY-MM-DD HH:MI:SS。

## HTML元素

#### A(class,text, href)

该函数创建一个具有指定类集（类）的* text *容器

#### Div(class,text)

该函数创建具有指定类集（类）的

text </ div>容器

#### Divs(class,[class,]) ... DivsEnd

该函数将页面模板的一部分包含在彼此包围的div容器中; 容器的数量等于指定类的参数编号。

```
Divs(md-5, panel)
```

```
    Any content.
DivsEnd:
```

**P(class, text)**

该函数创建一个具有指定类集（类）的* text *容器。

**Em(class,text)**

该函数创建具有指定类集（类）的* *text* *容器。

**Small(class,text)**

该函数使用指定的类（类）创建一个* text *容器。

**Span(class,text)**

该函数创建具有指定类集（类）的* text * </ span>容器。

**Strong(class,text)**

该函数创建具有指定类集（类）的**** text **** 容器。

**Label(text,[class])**

该函数使用指定的类创建一个* text * 容器。 如果语言表中包含文本中指示的值的语言资源，则文本将自动翻译和转换。

**Legend(class, text)**

该函数使用指定的类（lasslass）创建一个
* text *
容器。

**Tag(tagname, [text], [class])**

该函数创建具有指定类的* text * 容器; 支持h1-h6及按钮标签。

**Image(src, [alt], [class] )**

该函数将图像插入页面。

- src - 图像源指示符;
- alt - 鼠标移动到图像显示的文字;
- class - classes的列表。

#### MarkDown(text)

该函数将转换为具有markdown布局的HTML文本。 例如,

```
MarkDown(`## Header
      Any Text
`)
```

#### Val(idname)

该函数通过其标识符（id）返回HTML元素的值。

## 条件结构

#### If(condition) ... Else ... ElseIf ... IfEnd

条件构造允许根据条件表达式的真实性或虚假性显示页面模板的不同片段。 如果构造可能是封闭的，例如,

```
If(#value#)
    Divs(myclass)
        If(#par#)
           ...
        IfEnd:
    DivsEnd:
ElseIf(#param2#)
    P(class, Text)
Else:
    Divs(myclass2)
        .....
    DivsEnd:
IfEnd:
```

## 显示表单元素

## Form(class) ... FormEnd

该函数将页面模板的一部分与带有指定类集（类）的

... </ form>容器相结合。

#### Input(idname,[class],[placeholder],[type],[value])

该函数创建一个表单输入字段。

- idname - 字段标识符名称;
- class - 类的列表;
- placeholder - 提示文字;
- type - 字段类型，默认为text;
- value - 默认值。

#### Textarea(idname,[class],[value])

该函数显示textarea类型表单字段。

- idname - 字段标识符名称;
- class - class的列表,
- value - 默认值。

#### InputAddress(idname,[class],[value] )

该函数创建一个用于输入钱包地址的表单域，输入地址时，建议的选项显示在下拉列表中。

- idname - 字段标识符名称;
- class - class的列表,
- value - 默认值。

#### InputDate(idname,[class],[value] )

该函数创建日期和时间输入的表单域。

- idname - 字段标识符名称;
- class - class的列表,
- value - 默认值。

#### InputMoney(idname,[class],[value])

该函数创建用于输入货币值的表单域。

- idname - 字段标识符名称;
- class - class的列表,
- value - 默认值。

**Select(idname, list, [class], [value])**

该函数创建一个 < select > 下拉列表。

- idname - 标识符;
- list - 传输值列表;
- value - 默认选择的列表值;
- class - class的列表。

定义的 *list* 有2个选项:

1. 以逗号列出列表名称,
2. 从 tablename.column.idname 格式的数据库表中获取值，其中 tablename 是表名，column是列名称，其值显示为列表名称，idname - 列的名称，其名称值用作

**TextHidden(idname,....)**

该函数创建了大量隐藏的textarea字段; 用逗号分隔的名称设置为标识符（id）; 字段值取自具有相同名称的变量。 例如，如果变量 #test# = “Line” 存在，则TextHidden(test) 将创建一个带有 id=”test” 和 “Line”值的textarea。

**Source(idname,[value])**

该函数显示文本输入字段，其中突出显示运算符、关键字等。例如，用于编辑合约

- idname - 标识符;
- value - 默认值。

## 从数据库获取值

**ValueById(table,idval,columns,[aliases])**

该函数将数据库表项中的值，通过字符串的id值获取。

- table - 表名;
- idval - 获取的条目的id值;

- columns - 以逗号分隔的列的名称；默认情况下将创建具有列名称的变量，接收到的值将被传输到该变量；
- aliase - 列名称之外的变量名称，以逗号分隔，与列名称相同。

例如，*ValueById(#state_id#_citizens, #citizen#, “name,avatar”, “FirstName,Image”)

**GetList(name, table, colnames, [where], [order], [limit])**

该函数从表中获取条目。

- name - 使用 ListVal 或 ForList 函数，从结果列表中提取特定记录的名称；
- colnames - 由逗号分隔的结果列表;；应首先指定具有索引的列，并通过此值访问值 ListVal 和 ForList 将被实现；
- where，order，limit - 条件，排序和结果字符串数。

**ListVal(name, index, column])**

该函数返回从**GetList**函数获取的列表中的值。

- name - GetList 函数中指定的名称将用作参数值；
- index - 第一列搜索标识符的值，如GetList所示；
- column - 具有返回值的列的名称。

**ForList(name) ... FormListEnd**

该函数显示使用**GetList**函数获得的条目的完整列表；*GetList*函数中指定的名称将用作名称参数值。一个记录显示模板的结尾由**FormListEnd**关闭功能修复。条目列的值包含 #name_column# 类型的变量，其中表列的名称在下划线之后指示；#index# 变量可用，其中包含条目的序列号，以1开头。

```
GetList(my, #state#_mytable, "id,param,value")
ForList(my)
    Divs(md-5, panel)
       Strong(#my_index#: #my_ param #)
       P(pclass, #my_value#)
    DivsEnd:
ForListEnd:
```

**GetOne(colname, table, where, [value])**

该函数根据条件从数据库表中返回值。

- colname - 返回列的名称;
- table - 表的全名（#state#_mytabl）;
- where - 条件;
- value - 条件值，如果没有指定value参数，那么where参数将包含完整的查询。

**GetRow(prefix, table, colname, [value])**

该函数通过根据指定的字段和值，或根据请求搜索获得的数据库表项中的值，形成一组变量。

- prefix - 用于形成其中写入所生成条目的值的变量名称前缀：变量具有以下格式：#prefix_id#、#prefix_name#，其中在下划线后指定表列的名称;
- table - 表的全名（#state#_mytable）;
- colname - 搜索条目的列名称;
- value - 搜索条目的值，如果未指定值参数，则colname参数应包含对表的完整“where”查询。

**StateVal(name, [index])**

该函数从state_parameters表中显示参数值。

- name - 值名称;
- index - 值序列号，如果它们的列表用逗号表示，例如*gender | male,femal*，则 StateValue(gender, 2) 将返回 *femal*;

如果具有生成名称的语言资源可用，则其值将被替换。

**Table**

该函数使用数据库中的值创建一个表。 该函数具有命名参数，它们显示在形状按钮中：

- Table - 表名;
- Order - 排序表行的列名称，可选参数;
- Where - 采样条件，可选参数;
- Columns - 显示列数组，由标题和值 [[ColumnTitle, value],...] 组成;；对应于该行的基表中的列值作为具有列名称 (#column_name#) 的变量返回。

## 显示合约

**BtnContract(contract, name, message, params, [class], [onsuccess], [pageparams])**

该函数创建一个按钮，当单击时，打开一个模态窗口，提示拒绝或确认合约的显示。

- contract - 合约名称;

- name - 列名称;
- message - 模态窗口的文本;
- param - 转入合约的参数;
- class - 按钮类列表;
- onsuccess - 如果合约成功执行，应该进行转移的页面名称;
- pageparams - 以逗号分隔的以 var:value 格式转移到页面的参数。

例如，*BtnContract(DelContract, Delete, Delete Item?, "IdItem:id_item","btn btn-default')*

**TxButton**

该函数创建一个按钮，当单击启动合同时执行。该函数具有命名参数，显示在形状按钮中:

- Contract - 合约名称;
- Name - 按钮名称，默认发送;
- Class -
  容器的类列表，带有按钮;
- ClassBtn - 按钮的类列表;
- Inputs - 转移到合约的值列表。默认情况下，合约参数值（数据部分）取自具有相同标识符（id）的HTML元素（例如，表单域）。如果元素标识符与合约参数名称不同，则使用输入"contractField1=idname1, contractField2=idname2" 格式中的赋值。可以分配格式的变量值："contractField1#=var1, contractField2=var2" （变量 #var1# 和 #var2# 的值将被转移）;
- OnSuccess - 成功执行合约时将进行转移的页面名称，并将格式为 var:value 的参数传输到以逗号分隔的页面，例如 "CompanyDetails, CompanyId:#CompanyId#" ;
- Silent - 在值为1的情况下 - 在成功完成合约时显示信息; *自动关闭 - 如果值为1，则在成功完成合同后自动关闭消息。

例如,

```
TxButton {
    Contract: MyContract,
    Inputs: 'Name=myname, Request #= myreq',
   OnSuccess: "MyPage, RequestId:# myreq#"
}
```

**TxForm**

该函数创建一个用于输入合同数据的表单。 该函数具有命名参数，它们显示在形状按钮中:

- Contract - 合约名称;
- OnSuccess - 在成功执行合约的情况下，将进行移转的页面名称，并将格式为 var:value 的参数传输到页面，以逗号分隔，例如"CompanyDetails, CompanyId:#CompanyId#";

- Silent - 如果值为1，则显示合约成功完成的信息；
- AutoClose - 如果值为1 - 成功完成合约的消息将自动关闭。

```
TxForm {
    Contract: MyContract,
    OnSuccess: 'mypage'
}
```

## 导航元素

#### Navigation(params, ...)

该函数显示一个带有“面包屑”类型的位置导航面板，以及编辑当前页面编辑的链接。例如，导航（LiTemplate(dashboard_default, citizen),government）。

#### LinkPage(page,text,[params])

该函数创建一个按钮，当单击时，允许转移到指定的页面。如果未指定参数名称，文本将与页面相同。可以使用该函数创建到系统页面的链接。在这种情况下，在页面名称之前添加 sys-前缀，例如 LinkPage（sys-interface、Interface）。

- page - 页面名称；
- text - 链接文字；
- params - 以逗号分隔的以 var:value 格式转移到页面的参数。

#### LiTemplate(page, [text], [params], [class])

该函数创建包含页面的ling的

- * text * </ li>容器

- page - 页面名称；
- text - 链接文字；
- params - 以逗号分隔的以 var:value 格式转移到页面的参数；
- class - 类别清单。

```
LiTemplate(mypage, Home page, global:1)
```

**BtnPage(page, name,[params],[class], [anchor])**

该函数创建一个按钮，当单击时，允许转移到指定的页面。 如果没有指定类的参数，则按钮将具有btn btn-primary类。 可以使用该函数创建到系统页面的链接。 在这种情况下，请在页面名称之前添加sys-前缀。 例如，BtnTemplate（sys-interface、Interface）。

- page - 转移页面的名称;
- name - 列名称;
- params - 传送到页面的参数;
- class - 按钮类的列表;
- anchor - 一个锚（页面元素的id），将页面滚动到指定的位置。

**BtnEdit( page, icon, [params] )**

该函数以齿轮的形式创建一个指向指定页面链接的按钮，并将该ID作为参数传输; 它在屏幕表中用于引用元素编辑。 要进入系统页面或应用程序页面，您必须相应地添加系统和应用程序前缀。 例如，BtnEdit(sys-editPage, cog, "name: #name#, global: #global#")。

**Back(page, [params])**

该函数进入历史记录中指定页面的显示。

- page - 页名;
- params - 显示var:value格式的历史记录页面的参数，以逗号分隔。

## 格式化页面模板

**PageTitle(header) ... PageEnd()**

该函数捕获页面主体并创建一个面板，其标题在标题参数中指示。

**Title(text)**

该函数使用content-heading类创建一个标题。

**FullScreen(state)**

当状态参数等于1时，该函数将页面工作区域的宽度转换为窗口的整个宽度；当状态为0时，工作区域将缩小。

**WhiteMobileBg(state)**

该函数是移动设备全屏功能的模拟功能; 当状态参数等于1时，将页面工作区域的宽度转换为窗口的整个宽度，并在状态为0时缩小工作区域。

## 组织多级菜单

#### MenuItem(title, page, [params], [icon])

该函数创建一个菜单项。

- title - 菜单项的名称，如果在语言表中包含标题指示的值的语言资源，则文本将自动翻译和转换;
- page - 换页名称。 要进入系统页面，需要指定前缀sys-;
- params - 以逗号分隔的以var:value格式转移到页面的参数;
- icon - 一个图标。

#### MenuGroup(title,[idname],[icon]) ... MenuEnd:

该函数在菜单中形成一个子菜单。

- title - 菜单项的名称，如果在语言表中包含标题指示的值的语言资源，则文本将自动翻译和转换;
- idname - 子菜单标识符;
- icon - 一个图标。

```
MenuGroup(My Menu,mycitizen)
      MenuItem(Interface, sys-interface)
      MenuItem(Dahsboard, dashboard_default)
MenuEnd:
```

#### MenuBack(title, [page])

该函数将替换到父菜单（顶部菜单项）的链接。

- title - 菜单项的名称，如果在语言表中包含标题指示的值的语言资源，则文本将自动翻译和转换;
- page - 换页名称。

#### MenuPage(page)

该函数将页面参数中指定的页面设置为父菜单项。

# 数据表示

### Ring(count,[title],[size])

该函数显示中间带有count参数值的循环。

- title - 标题;
- size - 值大小。

### WiAccount(address)

该函数显示特殊设计的账号（钱包地址）传送到地址的参数。

### WiBalance(value, money)

该函数显示特殊设计货币格式的值，并添加货币参数中指定的货币名称。

### WiCitizen(name, address, [image], [flag])

该函数显示特殊设计的公民信息。

- name - 名称;
- address - 钱包地址，标准化为1234 -...- 5678的表格;
- name - 图像;
- name - 国旗。

### Map(coords)

该函数在页面上显示百度地图、腾讯地图或谷歌地图的容器，其坐标在coords参数中为{“center_point”:[“23.907173”,”54.333531”], “zoom”:”7”, “cords”:[[“23.915970”,”54.239502”], [“23.654588”,”55.371094”],[“22.958393”,”55.316162”]]} 。 容器高度取自预定义变量 #hmap# 的值（默认为100像素），宽度被拉伸到最大可能值。

### MapPoint(coords)

该函数在百度地图、腾讯地图或谷歌地图容器的页面上显示，在coords参数中指定的坐标上有一个标记。 容器高度取自预定义变量 #hmap# 的值（默认为100像素），宽度被拉伸到最大可能值。

### ChartPie

该函数显示圆形分格图表。该函数具有命名参数，它们显示在形状按钮中：

- Data - 以图表形式反映的数据 [[value,color,label],....]；每个列表项必须包含三个参数：value、rrggbb颜色和签名；如果此列表可用，则忽略其他参数；
- Columns - 用逗号分隔的rrggbb颜色列表；
- Table - 要从中获取数据的表的名称；
- FieldValue - 具有值的列名称；
- FieldLabel - 具有签名的列名称；
- Order - 排序表行的列名称，可选参数；
- Limit - 采样条件，可选参数。

#### ChartBar

该函数将图表显示为列。除Data之外的所有参数与 ChartPie 函数相同。

## 显示语言资源

#### LangJS(resname)

该函数创建具有语言资源值的 * resname * </ span>容器。 它用于自动替换浏览器中的语言资源。（这些是在 static/js/lang/*.js 中描述的资源）

#### LangRes(resname)

该函数从语言表中返回指定名称的语言资源。

## 服务函数

#### BlockInfo(blockid)

该函数显示具有区块编号（blockid）的链接，当单击时，将打开一个带有区块信息的窗口。

#### TxId(txname)

该函数返回指定的交易标识符。 例如，

```
SetVar(
type_new_page_id = TxId(NewPage),
type_new_contract_id = TxId(NewContract)
)
```

**Json(data)**

该函数使用jdata变量创建一个脚本标记，并以json格式分配data参数中指定的数据。 例如，

```
Json(`param1: 1, param2: "string"`)
```

我们会得到如下：

```
var jdata = { param1: 1, param2: "string"}
```

# api请求的描述

API允许根据请求创建私钥（钱包）接收资金，然后将其发送到其他钱包。 创建的所有私钥都以加密形式存储。

## 启动参数

为了使用此API，您需要在启动GAchain时指定以下参数。

-boltDir - 将创建并存储包含私钥文件的目录。 NoSQL数据库 BoltDB 用于存储密钥。 该文件名为exchangeapi.db。 如果未指定参数，则将在当前目录中创建该文件。

```
-boltDir=/home/temp
```

-boltPsw - 在写入数据库时加密私钥的密码。 如果在启动时未指定密码，API将无法正常工作。 首次启动时指定的密码应在随后的启动时指定。 切记不要忘记密码，因为密码不会在任何地方存储。 如果您指定的密码不正确或输入“console”一词作为密码，则在运行GAchain之后将提示您在控制台中输入密码。 此外，如果密码已经设置但未在命令行参数中指定，则将在控制台中请求该密码。

```
-boltPsw=mypass344
```

-apiToken - 此参数指定在向API发出请求时需要传递的代币（token）。 指定的代币（token）将被保存，并在以后的启动中被省略。 如果尚未指定此参数，则可以调用API命令而不指定代币

（token）参数。 如果您需要更改代币（token），您应该在此参数中启动一个新的值。

```
-apiToken=qweuytwuy347834
```

## API请求

对API请求的响应采用JSON格式，并且都含有一个错误字段。 如果该字段为空，则该请求已被执行且没有错误。 否则，该字段包含发生错误的文本。

### /exchangeapi/newkey

该命令生成私钥，将其记录到密钥文件中，并返回公钥和钱包地址。 例如：

*/exchange/newkey?token=qweuytwuy347834*

响应示例：

```
{"error":"", "public":"b7880fa40779d673e7ea026377d7182744869c081b1d3a1d613fe661333
ec67a74d985077555d80bc4aa65f5994f238def72881d6c2b6c60ffcc2ec7f050141d", "address":
"0773-5161-7272-4133-0241", "wallet_id":7735161727241330241}
```

### /exchange/send?sender=...&recipient=...&amount=...

该命令将钱从钱包（发送人）从数据库发送到指定的钱包（接收人）。 钱包可以为XXXX -...- XXXX，int64、uint64格式指定。 应该注意的是，该命令只发送交易，但不等待确认转账。 金额（amount）的值应在GAC中指明。

例如，

*/exchange/send?sender=1693-7869-8202-2463-0602&recipient=-3521799150320731671&amount=999000000000000
*

响应示例：

```
{"error":""}
```

### /exchangeapi/balance?wallet=....

该命令返回任意钱包的余额。

例如,

*/exchangeapi/balance?wallet=0773-5161-7272-4133-0241*

响应示例:


```
{"error":"","amount":"99992318000000000000","egs":"99.992318"}
```


**/exchangeapi/history?wallet=...&count=...**

该命令返回指定钱包中资金流的最后历史记录。 count是可选参数，并确定要返回的记录数（返回1个）。 默认情况下，返回50个条目，最大数为200。

例如,

*/exchangeapi/history?wallet=1693-7869-8202-2463-0602&count=10*

响应示例:


```
{"error":"","history":[{"block_id":"118855","dif":"-0.001","amount":"9999231800000
0000000","egs":"99.992318","time":"03.05.2017 10:48:14"},{"block_id":"118855","dif
":"-0.001999","amount":"99993318000000000000","egs":"99.993318","time":"03.05.2017
 10:48:14"},{"block_id":"112283","dif":"-0.001","amount":"99995317000000000000","e
gs":"99.995317","time":"02.05.2017 18:28:24"}]}
```