Hosted by Pawket

2023-3-18

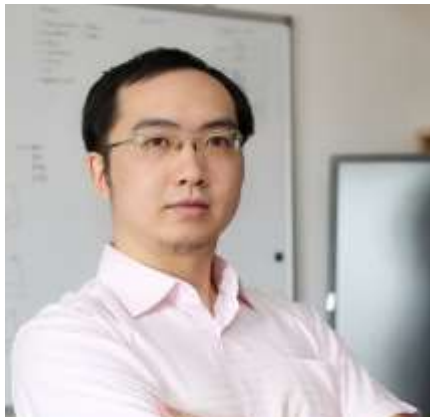
Chialisp开发入门

区块链开发工作坊

时间：2023年3月18日 14:00-17:30

地点：国康路100号上海国际设计中心22楼多功能厅

个人介绍

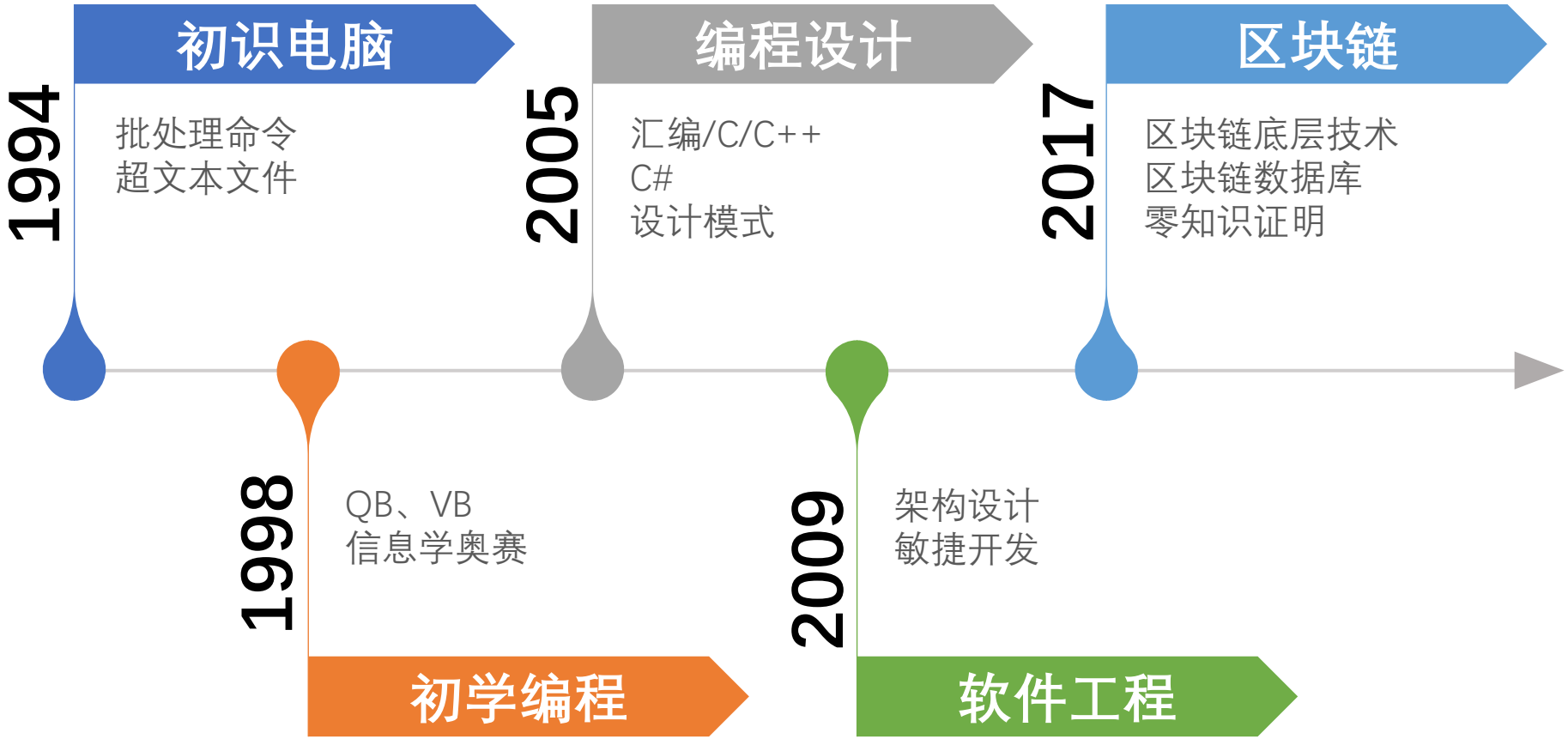


梁爽

区块链 架构师

上海交大 计算机博士生

微信：icerdesign
微博：@wizicer
Github: @wizicer
Twitter: @icerdesign
LinkedIn: www.linkedin.com/in/icerdesign

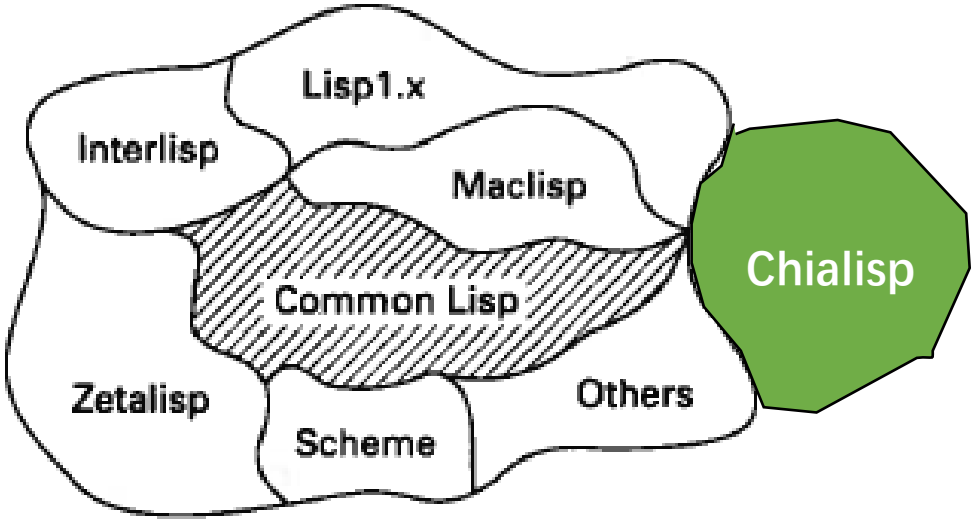


Lisp历史

- 由 John McCarthy 在 20世纪50 年代开发。
- 第二老的高级语言（第一是Fortran）
- LISP：List Processing Language 列表处理语言
  - **符号语言**：其执行效果可以用处理符号或数据串来表示
  - **函数式语言**：它的基本组成部分可以用数学函数表示
  - **通用语言**：它已经发展到包含许多“真实世界”的特征
  - **人工智能语言**：符号 <--> 概念、词

Lisp方言

- 方言很多
- Common Lisp是最标准的一个



(from “Common Lisp & Artificial Intelligence P. Harrison, Prentice Hall, 1990)

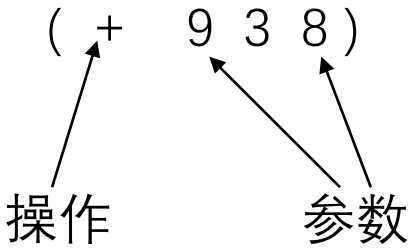
Lisp程序特点

- 实现特点：
  - 通常是解释型执行，这使得它变慢（存在编译器）
  - 内存密集型：大量的内存动态分配及释放
    - 有大量研究针对“垃圾收集”技术
  - 存在特殊的Lisp架构计算机，然而被现代CISC/RISC 架构取代了
- 程序特点：
  - 天然的符号特性，适合实验及原型（因此曾在AI 中流行）
  - 经常使用递归
  - 非常容易编写“一次性”的程序
    - 但必须加倍努力才能写出干净、政治正确的程序

Lisp程序

$$eval[e, a]$$

Lisp表达式　　变量赋值列表



LISP: “Lost in Stupid Parentheses”
（在一堆括号中迷路）

为什么用Lisp的人少?

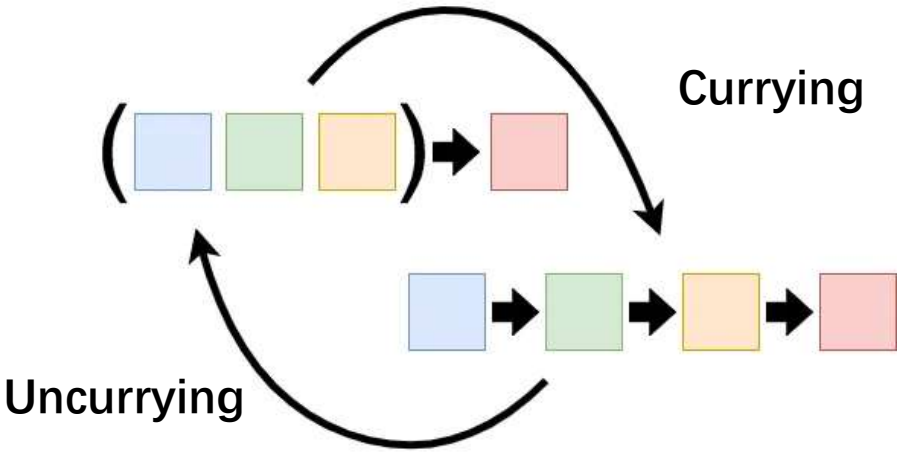
环境因素

- 可移植性/可用性
- 培训
- 流行度

门槛

- 副作用/纯代码
- 参考透明度
- 高阶函数
- Monad
- IO Monad
- Lambda算子
- 柯里化

Currying（柯里化）



p2_delegated_puzzle_or_hidden_puzzle.clvm

```
(mod (SYNTHETIC_PUBLIC_KEY original_public_key delegated_puzzle solution)
```

Puzzle

Fold | Copy Structure

(a (q 2 (q 2 (i 11 (q 2 (i (= 5 (point_add 11 (pubkey_for_exp (sha256 1 d77b699...8f26
()))))))) (q 2 23 47) (q 8)) 1) (q 4 (c 4 (c 5 (c (a 6 (c 2 (c 23 ())))) ())))) (a 23 47))) 1) (c (q 50 2 (i (i 5) (q 11 (q . 2) (a 6 (c 2 (c 9 ()))) (a 6 (c 2 (c 13 ()))))) (q 11 (q . 1) 5)) 1) 1)) (c (q . 0x924be539c4045a869b2c8934e3806760a92915d3edf4275110bccf6bde51a3fadecee86c8c 1))

p2_delegated_puzzle_or_hidden_puzzle

(a (q 2 (i 11 (q 2 (i (= 5 (point_add 11 (pubkey_for_exp (sha256 11 (a e9aaa49...fd52
2 23 47) (q 8)) 1) (q 4 (c 4 (c 5 (c (a 6 (c 2 (c 23 ())))) ())))) (a 23 47))) 1) (c (q 50 2 (i (i 5) (q 11 (q . 2) (a 6 (c 2 (c 9 ()))) (a 6 (c 2 (c 13 ()))))) (q 11 (q . 1) 5)) 1) 1))

synthetic_public_key

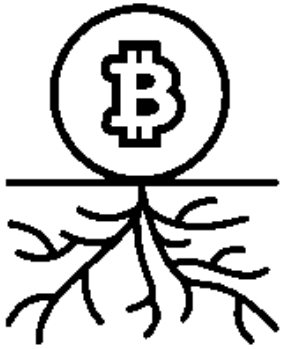
0x924be539c4045a869b2c8934e3806760a92915d3edf4275110bccf6bde51a3fadecee86c8c904

Solution Execute

```
()
(q
  (51 0xeff07522495060c066f66f32acc2a77e3a3e737aca8baea4d1a64ea4cdc13da9 1)
  (51 0xd2a26e38f80a966af8e5306fd2b2823353e91039feb52f8af8bc1abdfa8d8b3a 3)
  (60 0x493468e1a51ce81e1b36044c49c9d1a4e935b6680fbde595b9dace2d167953fb)
  (61 0xf03403e10cadab37225acf1d372d3c385826932cf4aa59753dbddc79368fd445))
())
```

为什么Chia选择Lisp

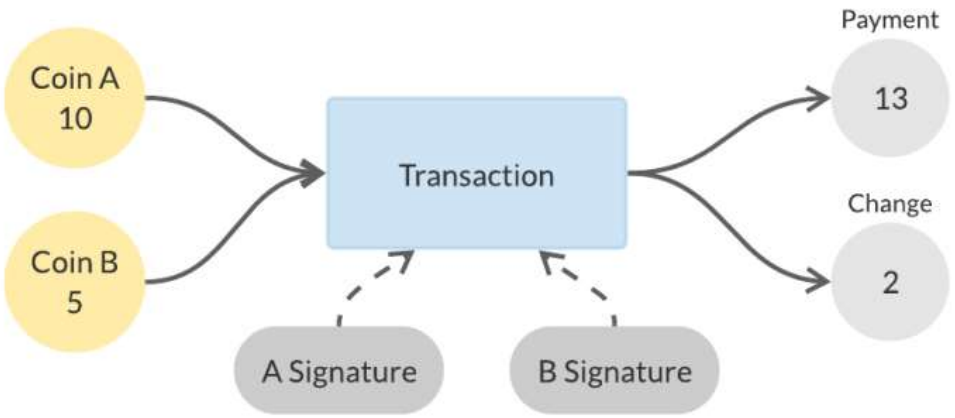
比特币 Taproot 激活

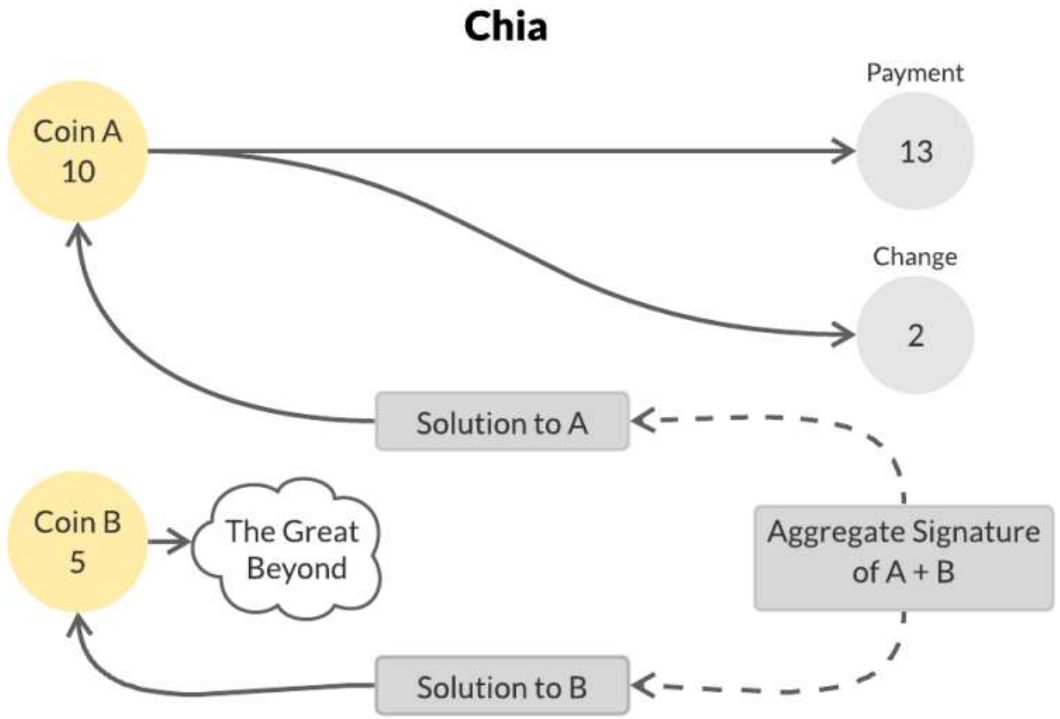


Taproot

- Schnorr Signatures (BIP 340)
- Taproot (BIP 341)
- Tapscript (BIP 342)

隔离见证

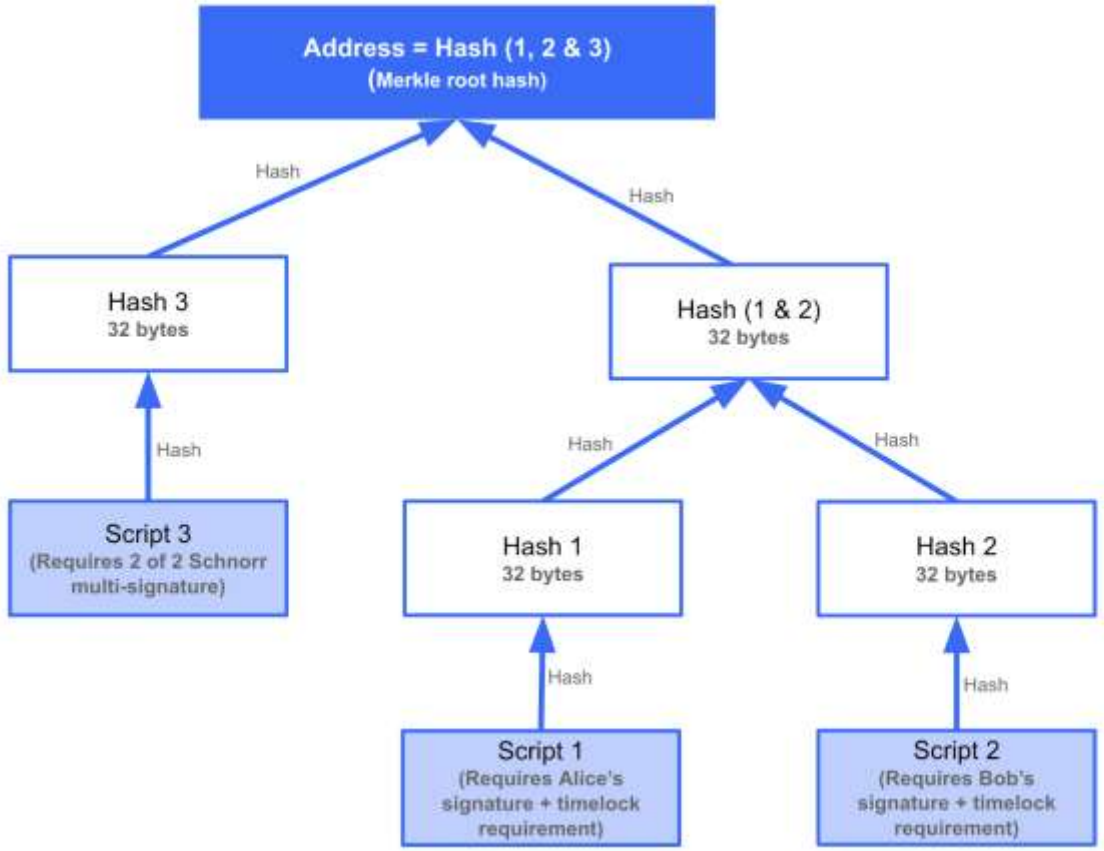


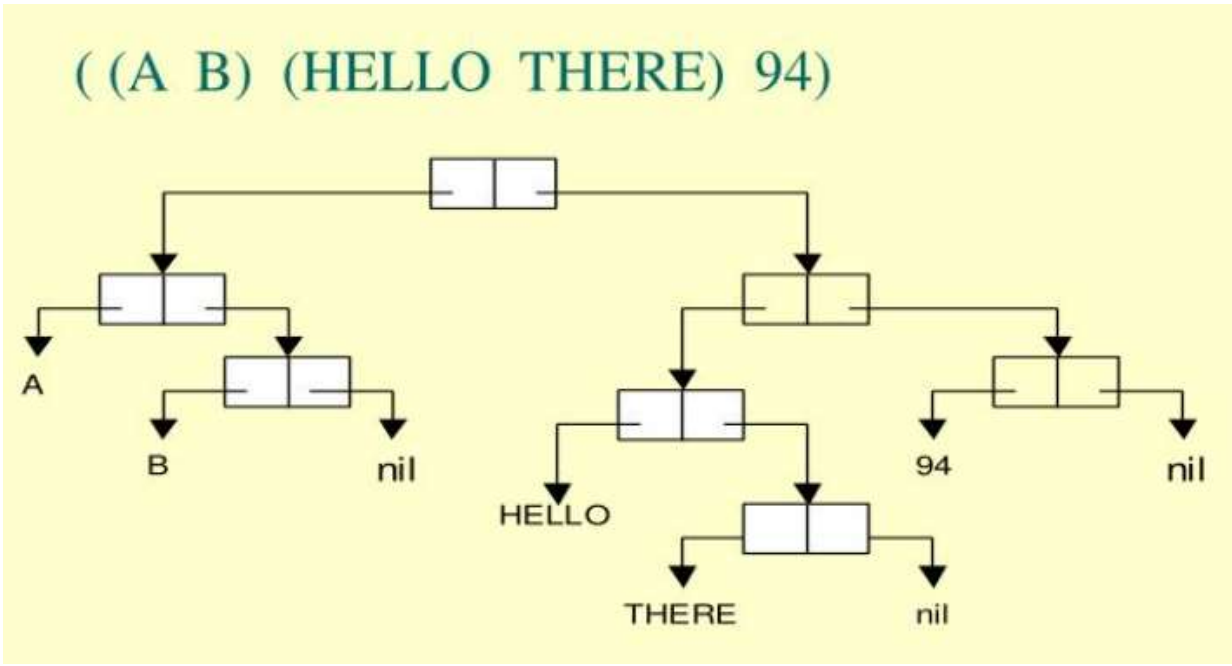


Schnorr Signatures (BIP 340)

https://www.chia.net/2021/05/27/agrgregated-sigs-taproot-graftroot/

TapScript





Merkelized Abstract Syntax Tree (MAST)

Tapscript (BIP 342)

https://blog.keys.casa/what-is-taproot-bitcoin-upgrade/

Chialisp 入门

Chialisp 是一种功能强大且安全的 LISP 类语言，用于通过智能合约功能来限制和释放资金。该网站是了解 Chialisp、CLVM 和条件语言的综合场所。

这是一个示例：

```
(mod (password new_puzhash amount)
  (defconstant CREATE_COIN 51)

  (if (= (sha256 password) (q . 0x2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7
    (list (list CREATE_COIN new_puzhash amount))
    (x)
  )
)
```

https://chialisp.chiabee.net

原子

原子是一串字节。这些字节可以解释为有符号大端整数和字节字符串，具体取决于使用它的运算符。

CLVM 中的所有原子都是不可变的。所有对原子执行计算的运算符都会为结果创建新原子。

原子可以用三种不同的方式打印，十进制、十六进制和字符串。十六进制值以 `0x` 为前缀，字符串用 `"` 引用。整数的打印方式不影响其底层值。十进制打印的原子 `100` 与十六进制打印的 `0x64` 相同。同样，值 `0x68656c6c6f` 与 `hello`。

将原子解释为整数时，重要的是要记住它们是有符号的。为了表示一个正整数，最高有效位可能不被设置。因此，正整数前面有一个 0 字节，以防设置下一个字节中的最高有效位。

Cons 结构

Cons 结构表示为一个括号，其中两个元素以 `.` 分隔。例如：

```
(200 . "hello")

("hello" . ("world" . "!!!"))
```

是合法的 cons 结构，但以下不是。

```
(200 . 300 . 400)
```

cons 结构总是有两个元素。但是，我们可以将 cons 结构链接在一起以构建列表。

Lists

list 用括号括起来，list 中的每个条目都是单行的，值之间没有句点。列表比 cons 结构更常用，因为它们更通用。

```
(200 300 "hello" "world")
```

您还可以嵌套 list。

```
("hello" ("nested" "list") ("world"))
```

请记住，list 是以空原子 () 结尾的连续 cons 结构的表示。以下表达式是相等的：

```
(200 . (300 . (400 . ())))

(200 300 400)
```

引用

要将原子解释为值而不是程序，需要用 `q` 引用它。引用的值形成一个 cons 结构，其中第一项是 `q` 运算符。例如，这个程序只是值 `100`：

```
(q . 100)
```

请注意，在更高级别的 Chialisp 语言中，不需要引用值。

list 和程序

list 是括号内的一个或多个元素的任何以空格分隔的有序组。例如：`(70 80 90 100)`、`(0xf00dbabe 48 "hello")` 和 `(90)` 都是有效 list。

列表甚至可以包含其他列表，例如 `("list" "list" ("sublist" "sublist" ("sub-sublist")) "list")`。

程序是可以使用 CLVM 评估的列表子集。**一个程序实际上只是一个用波兰表示法的列表。**

为了使列表成为有效程序：

- **1. list 中的第一项必须是有效的运算符**
- **2. 第一个之后的每个项目都必须是一个有效的程序**

规则 2 是文字值和非程序 list *必须* 使用 `q .` 引用的原因。

```
$ brun '(q . (80 90 100))'
(80 90 100)
```

现在我们知道我们可以在程序中放置程序，我们可以创建程序，例如：

```
$ brun '(i (= (q . 50) (q . 50)) (+ (q . 40) (q . 30)) (q
70
```

（更多关于后面使用的运算符）

CLVM 中的程序倾向于以这种方式构建。较小的程序组合在一起以创建更大的程序。建议您在括号匹配的编辑器中创建您的程序！

list 运算符

`f` 返回传递列表中的第一个元素。

```
$ brun '(f (q . (80 90 100)))'
80
```

`r` 返回列表中除第一个元素之外的所有元素。

```
$ brun '(r (q . (80 90 100)))'
(90 100)
```

`c` 将一个元素添加到列表中

```
$ brun '(c (q . 70) (q . (80 90 100)))'
(70 80 90 100)
```

我们可以使用这些组合来访问或替换列表中我们想要的任何元素：

```
$ brun '(c (q . 100) (r (q . (60 110 120))))'
(100 110 120)

$ brun '(f (r (r (q . (100 110 120 130 140)))))'
120
```

数学

CLVM 中不支持浮点数，仅支持整数。 CLVM 中的整数没有硬性大小限制。也支持负值。

数学运算符是`+`、`-`、`*` 和`/`。

```
$ brun '(- (q . 6) (q . 5))'
1

$ brun '(* (q . 2) (q . 4) (q . 5))'
40

$ brun '(+ (q . 10) (q . 20) (q . 30) (q . 40))'
100

$ brun '(/ (q . 20) (q . 11))'
1
```

要使用十六进制数字，只需在它们前面加上`0x`。

```
$ brun '(+ (q . 0x000a) (q . 0x000b))'
21
```

最后的数学运算符是相等的，它的作用类似于其他语言中的 ==。

```
$ brun '(= (q . 5) (q . 6))'
()

$ brun '(= (q . 5) (q . 5))'
1
```

正如你在上面看到的，这种语言将一些数据解释为布尔值。

布尔值

在这种语言中，空列表 `()` 的计算结果为 `False`。任何其他值的计算结果为 `True`，尽管在内部 `True` 表示为 `1`。

```
$ brun '(= (q . 100) (q . 90))'
()

$ brun '(= (q . 100) (q . 100))'
1
```

此规则的例外是 `0`，因为 `0` 与 `()` 完全相同。

```
$ brun '(= (q . 0) ())'
1

$ brun '(+ (q . 70) ())'
70
```

流程控制

`i` 操作符采用 `(i A B C)` 的形式，并作为一个 if 语句，如果 `A` 为真，则求值为 `B`，否则求值为 `C`。

```
$ brun '(i (q . 0) (q . 70) (q . 80))'
80

$ brun '(i (q . 1) (q . 70) (q . 80))'
70

$ brun '(i (q . 12) (q . 70) (q . 80))'
70

$ brun '(i () (q . 70) (q . 80))'
80
```

请注意，就像所有子表达式一样，`B` 和 `C` 都被急切地求值。 要将评估推迟到条件之后，必须引用 `B` 和 `C`（使用 `q`），然后使用 `(a)` 进行评估。

```
$ brun '(a (q . (i (q . 0) (q . (x (q . 1337) )) (q . (q
```

环境变量

到目前为止，我们的程序还没有任何输入或变量，但是 CLVM 确实支持这一点。

环境是传递给谜语的值列表。您只能传递一个 CLVM 对象作为环境，但该对象可以是任意长的列表。可以使用不带引号的 `1` 来引用整个环境。

```
$ brun '1' '("this" "is the" "environement")'
("this" "is the" "environment")

$ brun '(f 1)' '(80 90 100 110)'
80

$ brun '(r 1)' '(80 90 100 110)'
(90 100 110)
```

记住列表也可以嵌套。

```
$ brun '(f (f (r 1)))' '((70 80) (90 100) (110 120))'
90

$ brun '(f (f (r 1)))' '((70 80) ((91 92 93 94 95) 100) (
(91 92 93 94 95)
```

这些环境变量可以与所有其他运算符结合使用。

```
$ brun '(+ (f 1) (q . 5))' '(10)'
15

$ brun '(* (f 1) (f 1))' '(10)'
100
```

该程序检查第二个变量是否等于第一个变量的平方。

```
$ brun '(= (f (r 1)) (* (f 1) (f 1)))' '(5 25)'
1

$ brun '(= (f (r 1)) (* (f 1) (f 1)))' '(5 30)'
()
```

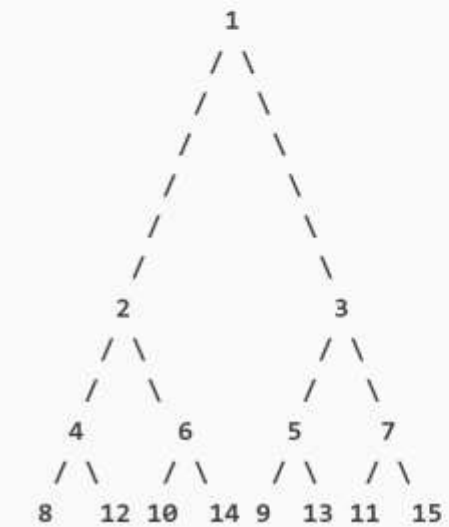
通过整数访问环境变量

在上面的例子中，我们只使用了 `1`，它访问树的根并返回整个谜底列表。

```
$ brun '1' '("example" "data" "for" "test")'
("example" "data" "for" "test")
```

但是，低级语言中的每个未加引号的整数都指代谜底的一部分。

你可以想象一个由 `f` 和 `r` 组成的二叉树，其中每个节点都有编号：



etc.

```
$ brun '2' '("example" "data" "for" "test")'
"example"

$ brun '3' '("example" "data" "for" "test")'
("data" "for" "test")

$ brun '5' '("example" "data" "for" "test")'
"data"
```

这被设计为在列表中也有列表时工作。

```
$ brun '4' '(("deeper" "example") "data" "for" "test")'
"deeper"

$ brun '5' '(("deeper" "example") "data" "for" "test")'
"data"

$ brun '6' '(("deeper" "example") "data" "for" "test")'
("example")
```

谜语和谜底

当我们在区块链上的硬币上下文中提及 Chialisp 时，我们将程序称为**谜语**，我们将环境或参数称为**谜底**。

```
brun <puzzle> <solution>
```

每当您想在 Chia 上花费一枚硬币时，您必须揭示它的谜语以及您想用来运行该谜语的谜底。如果谜语运行没有任何错误并返回一个有效的**条件**列表（更多关于下面的条件），则支出成功并处理条件列表。

币

一个币的主体由 3 条信息组成。这是定义币的实际代码：

```
class Coin:
    parent_coin_info: bytes32
    puzzle_hash: bytes32
    amount: uint64
```

1. 其父 ID
2. 谜语的树哈希（又名谜语哈希）
3. 它的价值

要构造一个币的 ID，只需将这 3 条信息按顺序连接起来的哈希值即可。

```
coinID == sha256(parent_ID + puzzlehash + amount)
```

这意味着币的谜语和金额是它的固有部分。您不能更改币的谜语或金额，您只能花费一个币。

花费

当你花掉一个币时，你就摧毁了它。除非谜语的行为指定在花费时如何处理硬币的价值，否则币的价值也会在花费中被破坏。

要花费一个币，您需要 3 条信息（以及可选的第 4 条信息）。

1. 币的 ID
2. 币谜语的完整源代码
3. 币谜语的解法
4. （可选）一组签名组合在一起，称为聚合签名

请记住，谜语和谜底与我们在基础知识中介绍的相同，只是谜语已经存储在币里面并且任何人都可以提交谜底。

网络没有硬币所有权的概念，任何人都可以尝试在网络上花费任何硬币。由谜语来防止硬币被盗或以意外方式花费。

如果有人可以提交硬币的谜底，您可能想知道某人如何"拥有"硬币。在指南的下一部分结束时，希望它应该是清楚的。

实践中的谜语和谜底

到目前为止，我们已经介绍了将评估某些结果的 ChiaLisp 程序。请记住，第一部分代表一个致力于锁定币的谜语，第二部分是任何人都可以提交的谜底：

```
$ brun '(+ 2 5)' '(40 50)'
90

$ brun '(c (q . 800) 1)' '("some data" 0xdeadbeef)'
(800 "some data" 0xdeadbeef)
```

这些是孤立的有趣练习，但这种格式可用于向区块链网络传达有关币在花费时应如何表现的指令。这可以通过将评估结果作为条件列表来完成。

条件

条件分为两类："*此支出仅在 X*" 时有效，"*如果此支出有效则 X*"。

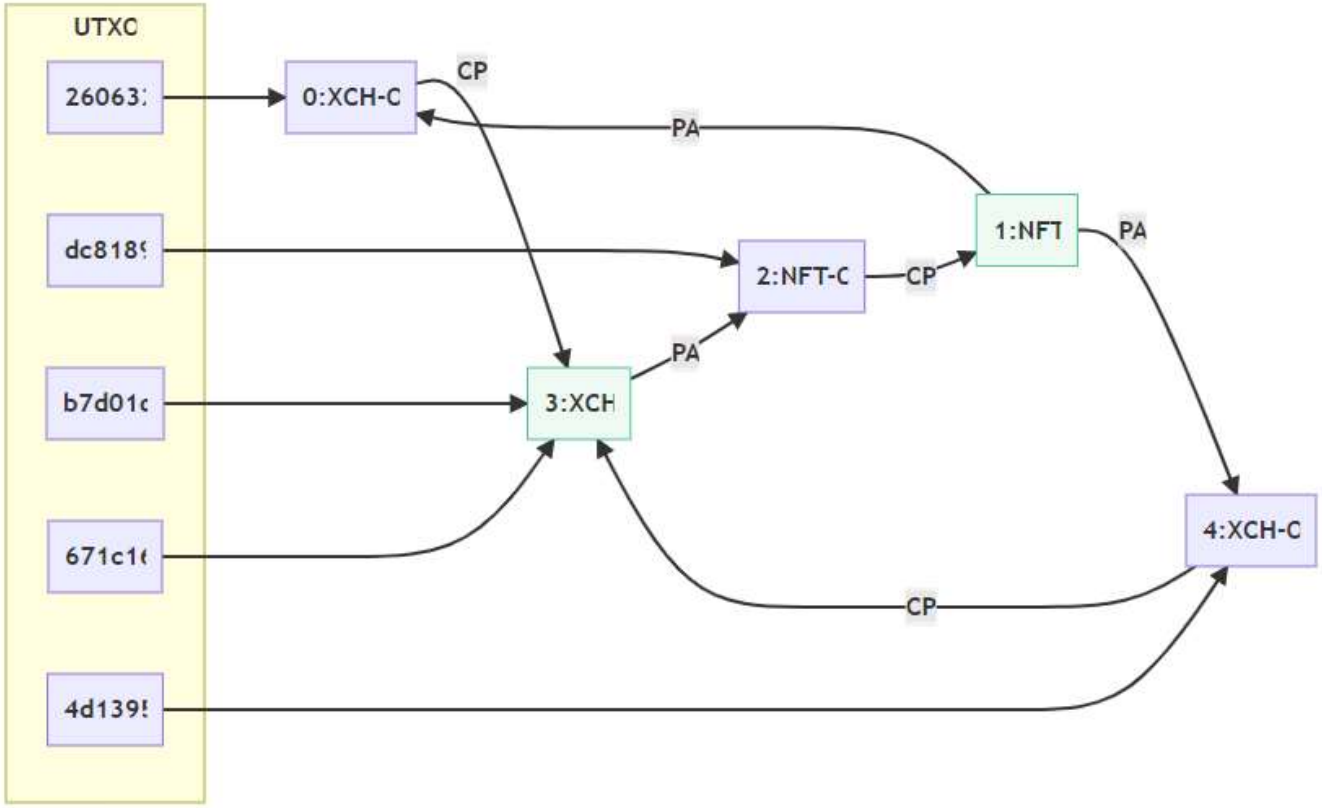
这是条件及其格式和行为的完整列表。

- **AGG_SIG_UNSAFE - [49] - (49 pubkey message)**：仅当附加的聚合签名包含来自给定消息的给定公钥的签名时，此支出才有效。这被标记为不安全，因为如果您对一条消息进行一次签名，则您拥有的任何其他需要该签名的硬币也可能会被解锁。由于硬币 ID 引入的自然熵，最好仅使用 AGG_SIG_ME。
- **AGG_SIG_ME - [50] - (50 pubkey message)**：仅当附加的聚合签名包含来自该消息的指定公钥的签名与硬币的 ID 和网络的创世挑战连接时，此支出才有效。
- **CREATE_COIN - [51] - (51 puzzlehash amount)**：如果此支出有效，则使用给定的谜语哈希和金额创建一个新硬币。
- **RESERVE_FEE - [52] - (52 amount)**：仅当本次交易中存在大于或等于*金额*的未使用价值（明确用作费用）时，此支出才有效。
- **CREATE_COIN_ANNOUNCEMENT - [60] - (60 message)**：如果此支出有效，则会创建一个临时公告，其 ID 取决于创建它的代币。其他币然后可以断言存在用于块内币间通信的公告。
- **ASSERT_COIN_ANNOUNCEMENT - [61] - (61 noticeID)**：只有在此区块中有与announcementID 匹配的公告时，此支出才有效。announcementID 是宣布消息的哈希值与宣布它的硬币的硬币 ID 连接起来`announcementID == sha256(coinID + message)`。
- **CREATE_PUZZLE_ANNOUNCEMENT - [62] - (62 message)**：如果此支出有效，则会创建一个临时公告，其 ID 取决于创建它的谜语。其他币然后可以断言存在用于块内币间通信的公告。
- **ASSERT_PUZZLE_ANNOUNCEMENT - [63] - (63 noticeID)**：只有在此区块中有与 announcementID 匹配的公告时，此支出才有效。 announcementID 是宣布的消息与宣布它的硬币的谜语哈希连接`announcementID == sha256(puzzle_hash + message)`。
- **ASSERT_MY_COIN_ID - [70] - (70 coinID)**：仅当提供的硬币 ID 与包含此谜语的硬币 ID 完全相同时，此支出才有效。
- **ASSERT_MY_PARENT_ID - [71] - (71 parentID)**：只有当提供的父代币信息与包含此谜语的代币的父代币信息完全相同时，此支出才有效。
- **ASSERT_MY_PUZZLEHASH - [72] - (72 puzzlehash)**：仅当提供的谜语哈希与包含此谜语的硬币的谜语哈希完全相同时，此支出才有效。
- **ASSERT_MY_AMOUNT - [73] - (73 amount)**：仅当显示的金额与包含此谜语的硬币的数量完全相同时，此支出才有效。
- **ASSERT_SECONDS_RELATIVE - [80] - (80 seconds)** ：此支出仅在自该硬币创建后经过给定时间后才有效。硬币的创建时间或"生日"由前一个块的时间戳定义，*而不是*创建它的实际块。类似
- **ASSERT_SECONDS_ABSOLUTE - [81] - (81 time)**：仅当此区块上的时间戳大于指定的时间戳时，此支出才有效。同样，硬币的生日和当前时间由前一个块的时间戳定义。
- **ASSERT_HEIGHT_RELATIVE - [82] - (82 block_age)**：此支出仅在自该硬币创建以来经过指定数量的区块时才有效。
- **ASSERT_HEIGHT_ABSOLUTE - [83] - (83 block_height)**：此支出仅在达到给定的 block_height 时才有效。

```
(
  (50
0xa4f7571bb6cbf55d5d26
eb 0x58379b23a09f35aeb
  (51 0xbae24162efbd568f
    (0xbae24162efbd568f8
  (51 0xbae24162efbd568f
  (51 0x167d41cf54d4d03
  (63 0xb38b006224027d2
```

- 50 AGG_SIG_ME
  - 0xa4f7571bb6cbf55d5d26db507e5b99daf66d9ba4fc37c6cc15c11d134b4824a7cfefdacb5079e6e94c842e3090be8feb
  - 0x58379b23a09f35aeb45f486babe6f3f15525a21a55e040683e2721713cb3e365
- 51 CREATE_COIN
  - 0xbae24162efbd568f89bc7a340798a6118df0189eb9e3f8697bcea27af99f8f79
  - 0x64
  - (0xbae24162efbd568f89bc7a340798a6118df0189eb9e3f8697bcea27af99f8f79)
  - amount: 100
  - Name: b7d01d84ef51aa4c6024d07b2a2d8796251aa842aa2192d8262e8936e00380fa
- 51 CREATE_COIN
  - 0xbae24162efbd568f89bc7a340798a6118df0189eb9e3f8697bcea27af99f8f79
  - 0x03
  - amount: 3
  - Name: 8d5deb6d8ee2920666995861e4994f1960a97cf5b48d20234334b115f54728c1
- 51 CREATE_COIN
  - 0x167d41cf54d4d03e317dd739fb80739d2bf96c67f635f1ce8051a28d471d696b
  - 0x2e
  - amount: 46
  - Name: 671c16b06f45b9be78bed9441b2d5ebb7d86fd97e07d3916ce74d0ddc0326fef
- 63 ASSERT_PUZZLE_ANNOUNCEMENT
  - 0xb38b006224027d2c4bbf86830641355b232d3cd783638626b8f6c16e3a1b7579

Mixch



Overall Check Check

Fee: 0

Puzzle Announcement

- 0 AS 1 0x8f0d29f322192b801abc32d47d7f66a96418871248086b76cc5d16bde14d1d31
- 1 No Assert 0x306dea26af8ccb89b4633d889eb0dbba3ae3bcd27a271d08ff934e90ea404acd
- 2 AS 3 0xb38b006224027d2c4bbf86830641355b232d3cd783638626b8f6c16e3a1b7579
- 4 AS 1 0x8d402b48779a5db48c2850f57f0e63c47d1f1491c3cdbf274f0e81ddd6f78b0e

Coin Availability

- 0 Ephemeral 0xb7d01d84ef51aa4c6024d07b2a2d8796251aa842aa2192d8262e8936e00380fa
- 1 Used 0x6c16b14afb634d46bf714ef05fe52c95acf5d0c73bd7fb90f3404c2bb2c26b92
- 2 Ephemeral 0x3075e7edc0a04491b3e8925d7f17ce188405a4cbed091362eaf4534a16b012a0
- 3 Used 0x0b0948f2215a0c8e811fe267bf1f1a4dbc034425423fba4fb08d4b5d9dc027d2
- 4 Ephemeral 0x8d5deb6d8ee2920666995861e4994f1960a97cf5b48d20234334b115f54728c1

Signatures Verified

- 1 0x6c16b...6b92 0x60f38...f622 0xa6393...1cfc
- 3 0x0b094...27d2 0x58379...e365 0xa4f75...8feb

https://mixch.dev/#/gallery

Chialisp SH Workshop

区块链开发工作坊

时间：3月18日 14:00-17:30

地点：国康路100号上海国际设计中心22楼多功能厅

2023 Mar 18th, 14:00-17:30

议程：

14:00-14:20 从B到C，区块链的发展史

14:20-15:00 Chia的技术路线图

15:00-15:40 丢币的N种方式（如何保障钱包安全？）

15:40-16:00 茶歇

16:00-16:40 Chialisp开发入门

16:40-17:00 Chia NFT介绍

17:00-17:30 现场提问时间