Maker协议分析

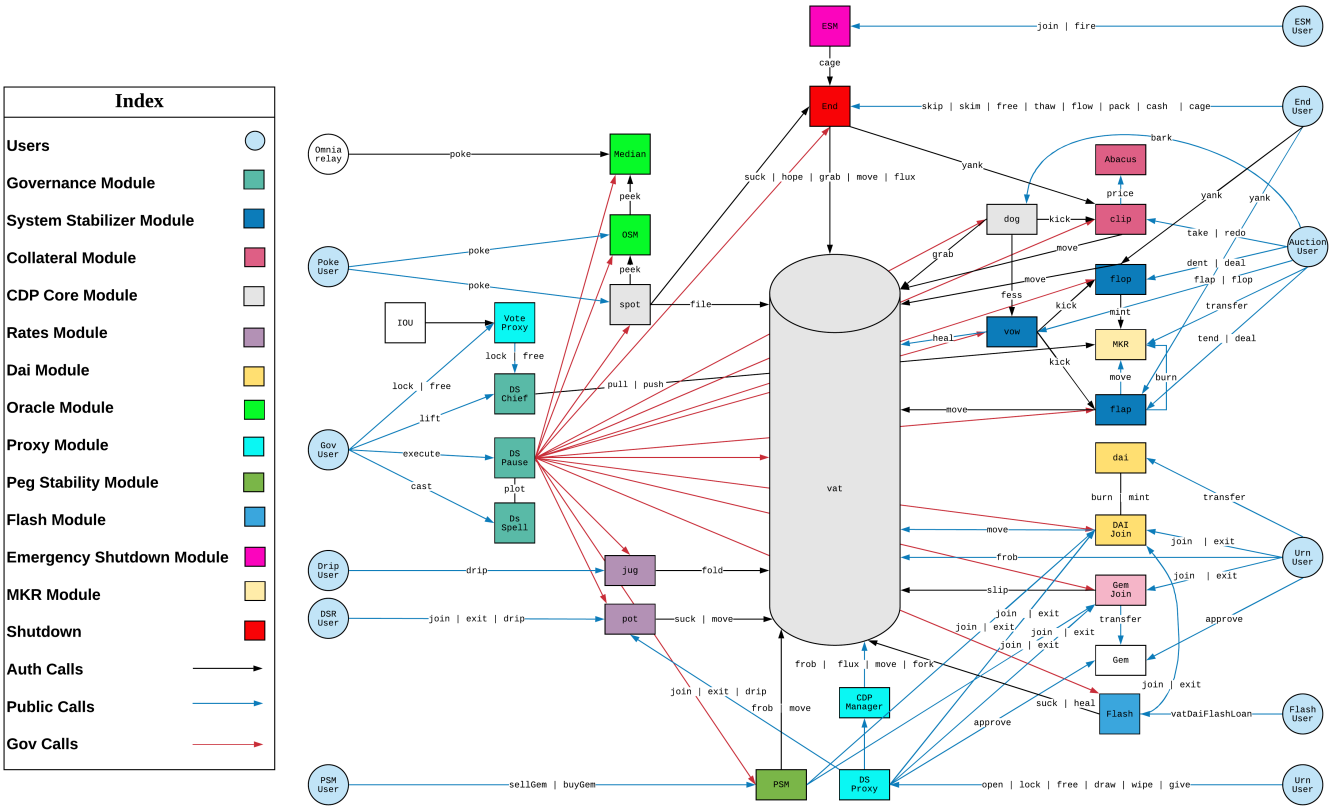
1. 概述

Maker协议是MakerDAO的多抵押Dai系统。代币Dai是锚定美元（价格 1:1）的稳定币。用户可以超额抵押借出Dai, 作为美元使用；也可以把Dai存入系统，获取存款利息。用户应用Dai的方式很简单，但是支撑Dai价值稳定的Maker协议还是相对复杂的。我们通过分析Maker协议的白皮书，技术文档以及智能合约，弄明白Maker协议的实现逻辑以及合约各个模块之间的关系，给出各个功能的实现以及合约之间的函数调用关系。通过这些，我们可以在Make协议基础上，开发自己的稳定币系统。

- **Maker协议**：Maker协议是一个平台，通过它，任何人可以用加密资产作为抵押物来生成Dai稳定币。Dai是一种以抵押物为后盾的稳定币，提供稳定性。MKR是一种治理代币，由利益相关者用来维护系统和管理Dai。
- **MakerDAO**：MakerDAO认为，一个去中心化的稳定币是任何企业或个人实现数字货币优势的必要条件。
- **多抵押物Dai**：多抵押物Dai（MCD）是Maker协议的新版本，已经在以太坊主网上发布和运行。MCD的最大变化是，它现在接受任何以太坊为基础的资产作为抵押物来生成Dai，只要它得到了MKR持有者的批准，并且通过Maker去中心化治理过程赋予了相应的风险参数。
- **MCD的新特性**：MCD还带来了一些新的特性，包括：
  - 支持多种金库抵押物类型（首发时为ETH和BAT）
  - 更强大的锚定机制（MKR作为最后防线）

2. 模块

下面是Maker协议中各个模块之间的关系.



2.1 core 模块

core模块是整个Maker协议的状态.

核心模块包含2个合约: vat, spot. 旧的文件以及图中把dog(cat), end也算入核心模块. 这里单独区分, dog属于清算模块, end 属于紧急关闭模块.

2.1.1 Vat合约: Vat合约是Maker协议的记账系统.

合约地址: 0x35D1b3F3D7966A1DFe207aa4514C12a259A0492B.

Vat合约是dss的核心金库(core vault)引擎，它存储了金库和所有相关的Dai和抵押物余额。它还定义了金库和余额可以被操作的规则。Vat合约中的规则是不可变的，所以从某种意义上说，Vat合约中的规则可以被视为dss的宪法。

2.1.1.1 核心原则:

- Dai不能没有抵押而存在

  ilk 表示一种抵押物, 比如usdt, eth, btc之类的; 在没锁定之前叫gem, 就是用户存进来, 但是还没借出Dai的时候.

  通过slip函数把gem添加进来, 通过flux把gem在用户之间转移.

- Vault的数据结构是Urn

Urn是一个结构体，用于表示特定类型的抵押品（collateral）在一个Vault中的信息。在Vat合约中，VaultDP都有一个对应的Urn结构体实例。以下是Urn结构体的字段及其含义：

1. ink: 这是Vault中锁定的抵押品数量（Locked Collateral），单位是wad。
2. art: 这是Vault的债务数量（Normalised Debt），单位是wad。这个债务是通过生成Dai来创建的。

- 抵押物的数据结构是Ilk

  Ilk是一个结构体，用于表示特定类型的抵押品（collateral）的信息。在Vat合约中，每种类型的抵押品都有一个对应的Ilk结构体实例。以下是Ilk结构体的字段及其含义：

  1. Art: 这是该类型抵押品对应的总债务（Total Normalised Debt），单位是wad。
  2. rate: 这是该类型抵押品的累积利率（Accumulated Rates），单位是ray。这个利率是通过系统的稳定性费用计算出来的。
  3. spot: 这是该类型抵押品的安全价格（Price with Safety Margin），单位是ray。这个价格是抵押品的市场价格乘以一个安全系数得到的。
  4. line: 这是该类型抵押品的债务上限（Debt Ceiling），单位是rad。这个上限是系统对该类型抵押品的总债务设定的最大值。
  5. dust: 这是该类型抵押品的最小债务（Urn Debt Floor），单位是rad。这个值是系统对单个Vault的债务设定的最小值。

2.1.1.2 Vault管理

- frob(bytes32 i, address u, address v, address w, int dink, int dart) external

  frob函数用于修改用户u的金库，使用用户v的代币作为抵押物，为用户w生成Dai。dink是改变的抵押品数量，dart是改变的稳定币数量。这个函数有一系列的条件检查，包括系统是否活跃，操作是否安全，操作是否被授权，以及债务是否超过上限等。

- grab(bytes32 i, address u, address v, address w, int dink, int dart) external auth

  grab函数用于没收用户u的金库，将代币给予用户v，为用户w生成sin。grab函数是清算金库的方式，将债务从金库转移到用户的sin余额。

- sin表示被没收或坏账，可以用等量的Dai来抵消。sin 是由 grab 和 suck 函数创建的，它们都会把 Vow 合约的地址作为 sin 的接收者。

- heal函数用于抵消sin和Dai。只有Vow合约才能成功调用heal。heal函数只能用正数作为参数，并会减少Dai和sin的余额。

- move函数用于在用户之间转移Dai余额。

2.1.1.3 fold 更新rate

function fold(bytes32 i, address u, int rate) external auth

- fold 函数用于更新某种抵押品类型（ilk）的 rate 参数，它是债务（art）和带有费用的当前价值之间的转换因子。
- old 函数的 rate 参数实际上是 Ilk.rate 值的变化量，即新旧比例因子之差。它是一个有符号整数，因此当前账户值可能增加或减少。
- Ilk.Art*rate 的数量会加到地址 u 的 dai 余额上，表示系统盈余的增加或减少；所有以指定 Ilk 抵押的 Vault 的债务余额会隐式地通过 Ilk.rate 加上 rate 来更新。

2.1.2 Spot合约:

合约地址: 0x65c79fcb50ca1594b025960e539ed7a9a6d434a3.

Maker协议中, Core合约和预言机(oracles)之间的联络员.

function poke(bytes32 ilk) external

这个函数首先从价格源(OSM)获取最新的价格，然后计算出新的清算价格，并将其存储在Vat合约中。最后，它触发一个Poke事件，通知外部监听器价格已经更新。

2.2 Dai 模块

Maker Protocol 中管理 DAI 代币和抵押品转换的模块。它包含了 DAI token contract 和多个 join adapters。

DAI token contract 是用户面向的 ERC20 合约，它维护了外部 DAI 余额的账目。它的大部分功能都是标准的可变供应代币，但它还有一个特别的功能，就是可以根据签名消息来授权转账(Permit)。

GemJoin合约是一系列的合约, 每种抵押物有不同的GemJoin.

- **GemJoin 合约**：它允许用户将标准的 ERC20 代币存入或取出 Vat 合约，以便与系统中的其他合约交互。它有一个 cage 函数，可以在系统出现问题时停止存入新的代币。

  join函数允许用户将ERC20代币加入到系统中作为抵押品.

  exit函数允许用户从系统中取出抵押品。

  GemJoin(ETH(WETH))地址: 0x2F0b23f53734252Bda2277357e97e1517d6B042A

GemJoin(BAT)地址: 0x3D0B1912B66114d4096F48A8CEe3A56C231772cA

- **DaiJoin 合约**：它管理了 Vat 合约中的 Dai 和 ERC-20 Dai 的转换。用户可以通过它将 Dai 提取为 ERC-20 代币(exit)，也可以将 ERC-20 代币销毁为 Vat 中的 Dai(join)。它也有一个 cage 函数，可以在系统出现问题时停止提取 Dai。

  DaiJoin合约地址: 0x9759A6Ac90977b93B58547b4A71c78317f391A28

2.4 清算2.0模块

Maker 协议的抵押物的拍卖行.

清算是当一个 Vault 的抵押率不足时，自动将其抵押品和债务转移给协议的过程。清算合约（Dog）会立即启动一个拍卖，出售抵押品换取Dai，试图抵消协议承担的债务。

Dog合约地址: 0x135954d155898D42C90D2a57824C690e0c7BEf1B

每种抵押品都对应一个Clipper合约和Abacus合约, 其中ETH是:

Clip(ETH): 0xc67963a226eddd77B91aD8c421630A1b0AdFF270

Abacus(ETH): 0x7d9f92DAa9254Bbd1f479DBE5058f74C2381A898

2.4.1 特点

- 立即拍卖

  拍卖合约使用了 Dutch (荷兰式)拍卖的方式，即根据初始价格和流逝时间计算出当前价格，并允许即时成交。这样可以减少竞拍者的资金锁定和价格波动风险，也可以提高资金回收速度。

- 抵押品闪电贷

  它使得竞拍者不需要有任何资金（除了燃料费）就可以参与拍卖。竞拍者可以通过将拍卖的抵押品在其他协议中换成 DAI 来购买抵押品。这样，整个 DeFi 中的所有 DAI 流动性都可以被任何竞拍者利用，只受燃料费的限制。

  它允许竞拍者指定一个合约(特定接口)地址和一些数据，拍卖合约会自动调用该合约中的逻辑。

- 价格时间函数

  指的是拍卖中的价格随着初始价格和设定时间的变化而变化。不同的抵押品类型可能需要不同的价格曲线，这还在研究中。目前有一些选项，如线性、阶梯指数和连续指数等.

指的是拍卖中的价格有时可能会上升，而不是一直下降。这可能导致竞拍者付出高于预期的价格。造成这种情况的原因可能是拍卖被重置（通过 redo）或者治理改变了价格计算器的参数。建议竞拍者（或者竞拍界面）仔细选择最高可接受价格（max）参数，以防止价格意外上涨。

- 重置拍卖

  这是一个过程，它用于处理拍卖因为时间过长或价格过低而失效的情况。拍卖重置的条件由 tail 和 cusp 两个参数控制。重置函数会检查拍卖是否需要重置，然后将拍卖的开始时间和价格重新设定为当前时间和 OSM 价格加上 buf 参数。这个过程会一直重复，直到拍卖完成或被取消。

- 提高Keeper钱包安全

  它使得竞拍者可以使用 clipperCallee 模式来参与拍卖，而不需要在自己的账户上存储 DAI 或抵押品。竞拍者只需要有足够的 ETH 来执行交易，将抵押品发送到一个合约，该合约会返回 DAI 给 msg.sender 来支付抵押品。该合约实现了 clipperCallee 接口，可以将多余的抵押品或 DAI 发送到一个冷钱包地址，该地址对竞拍者不可访问。

2.4.2 合约细节

Dog执行清算, Clipper执行拍卖, Abacus为拍卖提供价格.

2.4.2.1 Abacus 合约

包含方法price包含了几种计算拍卖价格的方法:

function price(uint256, uint256) external view returns (uint256);

输入初始价格和时间, 计算出价格.

2.4.2.2 Dog 合约

它负责监控 Vault(Vault) 的抵押率，并在抵押率低于设定的最低值时，触发清算流程。

```
function bark(bytes32 ilk, address urn, address kpr) external returns (uint256
id)
```

参数ilk（抵押品类型），urn（Vault地址）和kpr（清算奖励接收者地址）。

当Vault的债务和清算罚金超过全局或单个抵押品类型的DAI目标上限时，只清算VaultDP，否则Vault个CDP。如果清算后的拍卖中的DAI太少，不足以吸引拍卖者，那么函数会回退

Dog.bark函数是清算Vault的主要函数，函数的执行过程：

- 首先，Dog.bar函数会检查保险库的抵押率是否低于规定的水平，如果是，就继续执行，否则，就回退。
- 然后，Dog.bar函数会计算保险库的债务和罚金，并检查是否超过了全局或单个抵押品类型的清算限额（Hole或ilk.hole）。如果超过了，就只清算部分债务和抵押品，以保证不超过这两个参数。如果没有超过，就清算全部债务和抵押品。
- 接着，Dog.bar函数会将保险库的债务和罚金加到全局和单个抵押品类型的清算累计器（Dirt或ilk.dirt）中，并将保险库的债务推入债务队列（vow.sin）中, 通过调用val.grab函数实现。
- 然后，Dog.bar函数会调用对应的拍卖合约（Clipper）的kick函数，来启动一个荷兰式拍卖。拍卖合约会根据预设的参数和价格函数来计算拍卖的初始价格和降价速度，并将拍卖编号返回给Dog.bar函数。
- 最后，Dog.bar函数会触发一个Bark事件，来通知外部监听器有一个新的拍卖开始，并将拍卖编号作为返回值返回。

每次调用清算函数（Dog.bark）时，会从一个保险库中转移所有债务, 而不是部分. 这样做的原因是，因为新的拍卖方式允许部分购买抵押品，所以参与者可用的流动性不再限制他们参与拍卖的能力，因此应该尽量减少拍卖的总数。需要强调的是，出售抵押品不再有最低DAI流动性要求。

清算系统2.0设置了两个参数：Dog.Hole和ilk.hole。Hole是全局参数，表示所有活跃拍卖所需的DAI总额的上限。ilk.hole是每种抵押品类型（ilk）对应的参数，表示该类型活跃拍卖所需的DAI总额的上限。当一个保险库被清算时，如果加上它的债务和罚金后超过了Hole或ilk.hole，那么只有部分债务和抵押品会被转移给拍卖合约（Clipper），以保证不超过这两个参数。这样可以控制每次清算所产生的DAI需求量，避免对DAI市场造成过大冲击。

2.4.2.3 Clipper合约

合约特点:

- Clipper合约是清算系统2.0的一部分，它取代了之前的Flipper合约，使用了更高效和灵活的拍卖方式。
- Clipper合约可以根据不同的抵押品类型（ilk）设置不同的参数，如起拍价格的缓冲系数（buf）、拍卖重置的时间（tail）、拍卖重置的价格下降比例（cusp）、激励保管者的百分比（chip）和固定费用（tip）等。
- Clipper合约可以根据预设的价格函数（calc）来计算拍卖的初始价格和降价速度，并根据市场情况动态调整。
- Clipper合约可以支持部分购买抵押品，也就是说，参与者可以根据自己的意愿和能力来购买一定数量的抵押品，而不必等待拍卖结束或购买全部抵押品。
- Clipper合约可以与外部合约进行交互，例如，参与者可以在购买抵押品后调用一个外部合约来执行一些操作，如出售、借贷或交换等。

主要函数的作用的解释：

- kick：开始一个拍卖，将抵押率不足的保险库的抵押品和债务转移给合约，并根据抵押品的市场价值和一个缓冲系数计算出拍卖的起拍价。该函数还会给启动拍卖的人发放一定的激励奖励。
- redo：重置一个拍卖，当拍卖超过一定时间或者价格下降到一定比例时，可以重新计算拍卖的起拍价，并延长拍卖的时间。该函数也会给重置拍卖的人发放一定的激励奖励。
- take：购买一定数量的抵押品，根据当前时间和起拍价计算出当前价格，并用DAI支付相应的金额。该函数还可以调用一个外部合约来执行一些操作，如出售、借贷或交换抵押品。
- upchost：更新缓存的最低清算金额，该金额是保险库的最低债务和清算罚金的乘积，用于防止产生过小的剩余拍卖。
- yank：取消一个拍卖，只有在紧急停止或者治理行动时才能使用。该函数会将抵押品退还给启动拍卖的人，并从累计器中减去相应的债务。

```
function kick(
uint256 tab, // Debt [rad]
uint256 lot, // Collateral [wad]
address usr, // Address that will receive any leftover collateral
address kpr // Address that will receive incentives
) external auth lock isStopped(1) returns (uint256 id)
```

它用于开始一个拍卖。它需要有授权权限，并且信任调用者将抵押品转移到合约中。

- **参数说明**：函数接受四个参数，分别是：
  - tab：需要筹集的债务金额（以 DAI 为单位）[rad]
  - lot：需要出售的抵押品数量（以抵押品为单位）[wad]
  - usr：如果拍卖有剩余的抵押品，将接收该抵押品的地址
  - kpr：将接收激励奖励的地址
- **返回值**：函数返回一个拍卖的 ID（id），用于标识该拍卖。
- **逻辑说明**：函数的主要逻辑如下：
  - 验证输入参数是否有效，生成一个新的拍卖 ID，并将其添加到活跃拍卖数组中。
  - 将输入参数保存到拍卖结构体中，并设置拍卖的开始时间（tic）。
  - 根据当前的 OSM 价格（val）和初始价格增加因子（buf）计算出拍卖的初始价格（top），并保存到拍卖结构体中。
  - 如果设置了清算奖励百分比（chip）或清算奖励固定费用（tip），则从 Vow 合约中提取相应的 DAI 金额（coin），并发送给激励地址（kpr), 通过vat.suck实现。
  - 触发一个 Kick 事件，通知外部观察者有新的拍卖开始。

```
function redo(
        uint256 id,  // id of the auction to reset
        address kpr  // Address that will receive incentives
    ) external lock isStopped(2)
```

redo用于重置一个拍卖，当拍卖超过一定时间或者价格下降到一定比例时，可以重新计算拍卖的起拍价，并延长拍卖的时间。以下是对redo函数的详细解释：

- redo函数的参数有两个：id和kpr。id是拍卖的编号，kpr是清算奖励者的地址。
- redo函数的返回值是一个布尔值，表示重置是否成功。
- redo函数的执行过程如下：
  - 首先，redo函数会检查拍卖是否存在，是否已经结束，以及是否满足重置的条件。重置的条件有两个：一是拍卖已经超过了预设的时间（tail），二是拍卖的当前价格已经低于起拍价的一定比例（cusp）。如果不满足这些条件，redo函数会回退。
  - 然后，redo函数会根据抵押品的市场价值(getFeedPrice)和一个缓冲系数（buf）重新计算拍卖的起拍价，并将其赋值给拍卖的状态变量（sale.price）。
  - 接着，redo函数会更新拍卖的结束时间（sale.tic），将其设置为当前时间加上预设的时间（tail）。
  - 然后，redo函数会给重置拍卖的人发放一定的激励奖励。激励奖励由两部分组成：一是抵押品的一定百分比（chip），二是一个固定费用（tip）。这些奖励会从合约中扣除，并发送给重置拍卖的人(vat.suck)。
  - 最后，redo函数会触发一个Redo事件，来通知外部监听器有一个拍卖被重置，并将返回值设置为true。

```
   function take(
        uint256 id,           // Auction id
        uint256 amt,          // Upper limit on amount of collateral to buy
[wad]
        uint256 max,          // Maximum acceptable price (DAI / collateral)
[ray]
        address who,          // Receiver of collateral and external call
address
        bytes calldata data   // Data to pass in external call; if length 0, no
call is done
    ) external lock isStopped(3)
```

take函数的详细解释：

- take函数的参数有四个：id，amt，max，who，data。其中：
  - id是拍卖的编号。
  - amt是想要购买的抵押品的最大数量（单位为wad）。

    - max是接受的最高价格（单位为ray），即DAI和抵押品的比率。
    - who是接收抵押品的地址，也是可以调用外部合约的地址。
    - data是传递给外部合约的数据，如果长度为0，则不调用外部合约。
- take函数的返回值没有定义，但会触发一个Take事件，包含了拍卖的相关信息。
- take函数的执行过程如下：
    - 首先，take函数会检查拍卖是否存在，是否已经结束，以及是否被紧急停止或治理行动取消。如果不满足这些条件，take函数会回退。
    - 然后，take函数会根据当前时间和起拍价计算出当前价格，并根据amt和max确定实际要购买的抵押品的数量和需要支付的DAI的金额。如果amt或max过大，take函数会自动调整为合理的值，以保证不超过拍卖所需的DAI总额（tab），也不低于拍卖剩余的DAI最低限额（chost）。
    - 接着，take函数会将抵押品从合约中转移给who，并将DAI从who中转移给vow（收益地址）。同时，take函数会更新拍卖和清算系统中的相关状态变量，如tab（拍卖所需DAI总额），lot（拍卖剩余抵押品数量），Dirt（全局清算累计器），ilk.dirt（单个抵押品类型清算累计器）等。
    - 然后，take函数会检查data是否为空，如果不为空，则调用who作为外部合约，并传递data作为参数。这样可以让who在购买抵押品后执行一些操作，如出售、借贷或交换等。但是为了安全起见，take函数不允许who是vat或dog合约的地址。
    - 最后，take函数会触发一个Take事件，来通知外部监听器有人参与了拍卖，并返回。

2.4.3 清算机制

2.4.3.1 清算激励机制:

- 清算激励机制是为了鼓励清算者（keeper）及时清算抵押率不足的保险库（Vault），并从拍卖中获得一定的奖励。这个特性与清算系统1.2不同，因为在1.2中，清算者可以在拍卖中优先出价，而在2.0中，没有这样的优势。
- 清算激励机制的形式是，对于每种抵押品类型（ilk），清算者可以获得一个固定的DAI金额（tip）和一个与清算债务成比例的DAI金额（chip）。这两个参数都可以被设为零。这样的结构有以下的理由：
    - 按照抵押品类型设置奖励，可以给予最大的灵活性，不仅可以考虑每种抵押品类型的风险参数，如mat（抵押率）和chop（清算罚金），还可以考虑一些只适用于某些抵押品类型的市场情况。
    - 与清算债务成比例的奖励，是为了奖励清算者为系统降低风险，因为风险本身与不充分抵押的保险库的规模成正比。换句话说，清算一个规模是另一个两倍的保险库，相当于减少了两倍的坏账风险。因此，清算者应该获得相似的奖励。而且，系统可以负担更大规模的清算，因为清算罚金也与保险库的债务成正比。
    - 固定的奖励，可以用来补偿清算者的燃料费用（这是每个保险库都要花费的），或者让MKR持有者有效地支付清算者来清理一些不太有吸引力的小规模保险库。

2.4.3.2 清算断路器：

- 清算断路器是一种可以暂停或限制清算功能的机制，它可以在发生紧急情况或治理行动时保护协议和用户的利益。与LIQ-1.2不同，清算系统2.0中的清算断路器有四个阶段，分别是：
  - 清算启用（0）：这意味着断路器没有触发，协议可以正常清算新的或旧的保险库（Vault）。
  - 新清算禁用（1）：这意味着不能启动新的清算（Clipper.kick），但可以继续进行已经开始的拍卖。
  - 新清算和重置禁用（2）：这意味着不能启动新的清算（Clipper.kick），也不能重置已经到达价格或时间终点的拍卖（Clipper.redo）。
  - 清算禁用（3）：这意味着不能启动新的清算（Clipper.kick），也不能购买拍卖中的抵押品（Clipper.take），也不能重置拍卖（Clipper.redo）。
- 清算断路器可以通过一个ClipperMom合约来控制，它可以让治理者绕过GSM延迟来快速执行断路器操作。GSM延迟是一种安全机制，它要求治理者在修改某些参数之前等待一段时间，以防止恶意行为。

2.4.3.3 紧急关停

- 通过Clipper.yank和End.snip函数来取消或转移拍卖。
- Clipper.yank函数可以由授权者调用，用于从合约中移除一个拍卖，并将剩余的抵押品和债务退还给调用者。这个函数主要用于紧急停止或治理行动时。
- End.snip函数是End模块的一个新函数，用于在紧急停止触发时，调用Clipper.yank函数来取消所有正在进行的拍卖，并将抵押品和债务退还给保险库的所有者。需要注意的是，退还的债务已经包含了清算罚金。

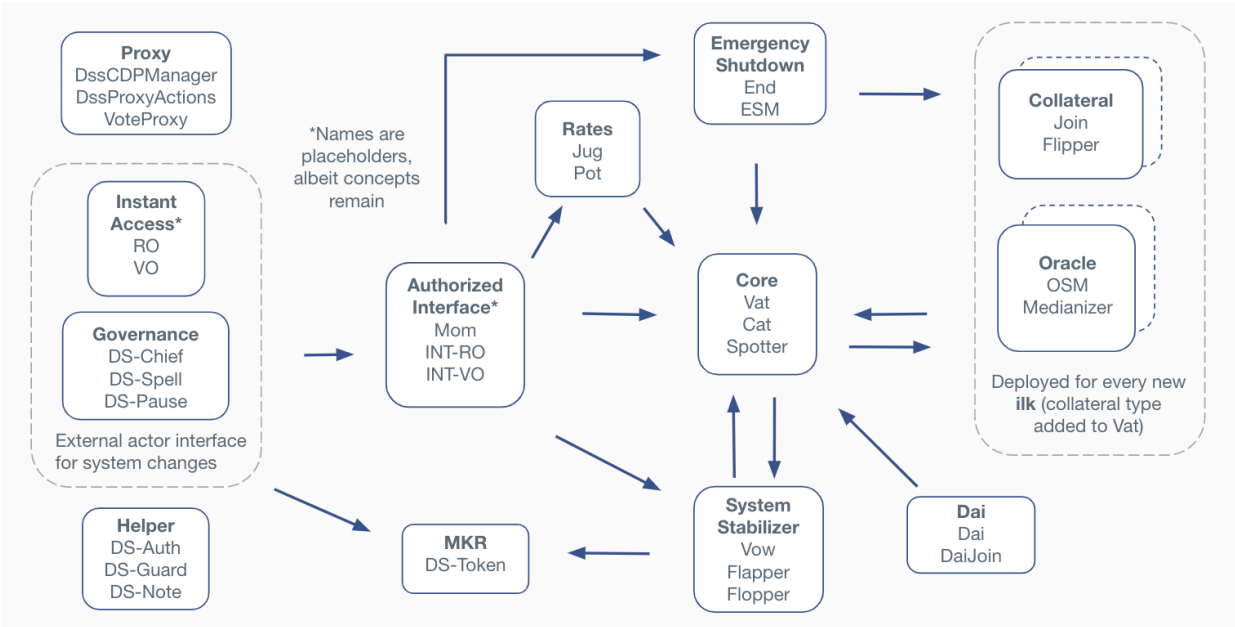
2.4.4 已知风险

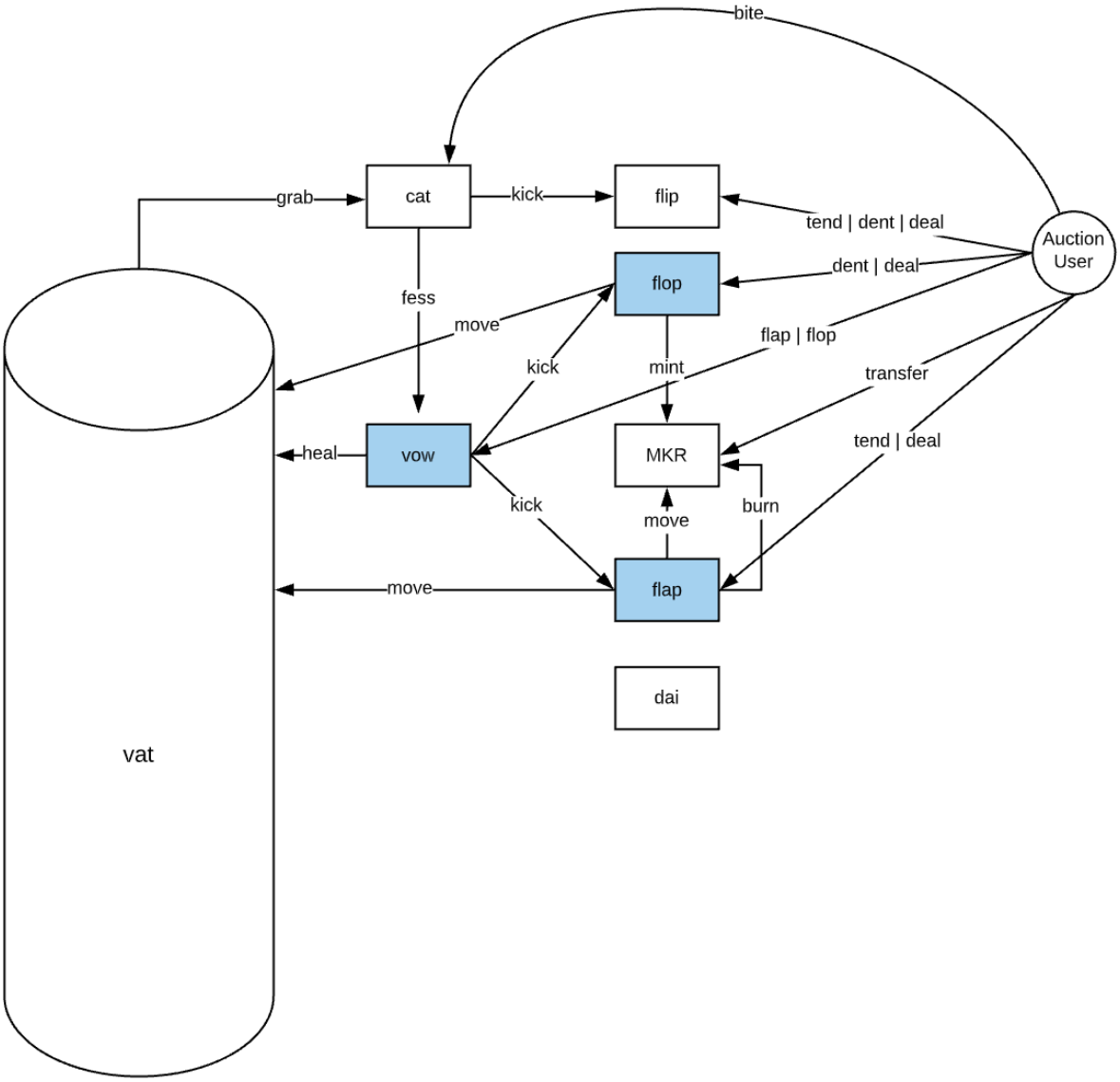
- 激励农场：如果治理者在提高最低债务或清算奖励时不谨慎，可能会诱使攻击者创建和清算大量的保险库来获取奖励，造成协议的损失。
- 价格下降过快：如果拍卖的价格下降得太快，可能会导致拍卖无人出价或结束，或者出价者支付得太低，造成拍卖的低效和坏账的产生。
- 价格下降过慢：如果拍卖的价格下降得太慢，可能会导致拍卖的价格始终低于市场价格，或者需要多次重置，造成拍卖的延迟和坏账的积累。
- 前运行风险：如果清算者使用无资本的方式参与拍卖，可能会面临一般性的前运行机器人的竞争，造成交易失败或损失。为了减轻这种风险，清算者应该使用授权代理合约，并提供一些自己的资本来出价。

2.5 系统稳定模块

保持Maker协议的稳定

- 清算系统模块的作用是在抵押品价值低于清算水平时，通过拍卖机制来恢复系统的稳定。
- 清算系统模块为拍卖参与者（外部行动者）提供了激励，让他们通过参与债务拍卖和盈余拍卖来帮助系统回到安全状态（系统平衡），并从中获得利润。
    - 清算系统模块有三个核心组件，分别是Vow、Flop和Flap合约。



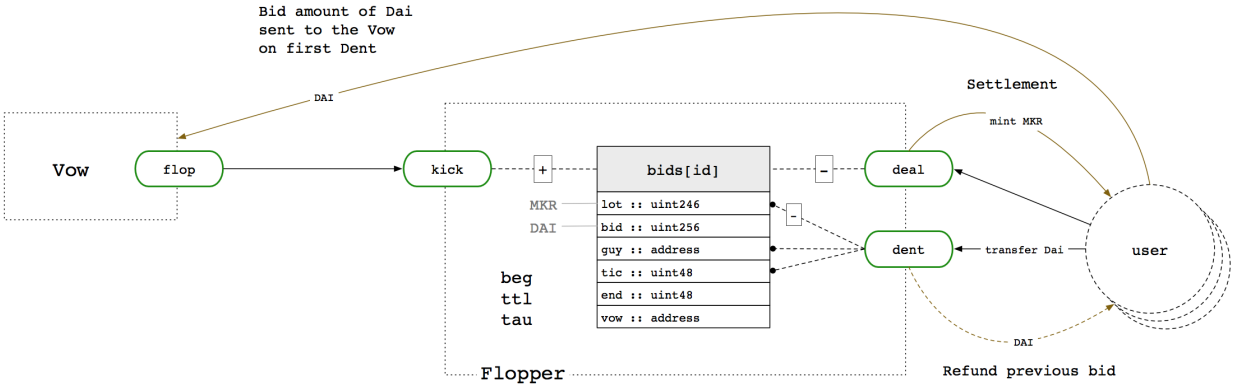


- Vow代表了整个Maker协议的平衡（包括系统盈余和系统债务）。Vow的作用是通过债务拍卖来弥补赤字，通过盈余拍卖来释放盈余。

  合约地址: 0xA950524441892A31ebddF91d3cEEFa04Bf454466

- Flop（债务拍卖）用于消除Vow的债务，通过拍卖新铸造的MKR来换取内部系统的DAI。拍卖结束后，Flop会将收到的DAI发送给Vow来抵消其债务，并为中标者铸造MKR。

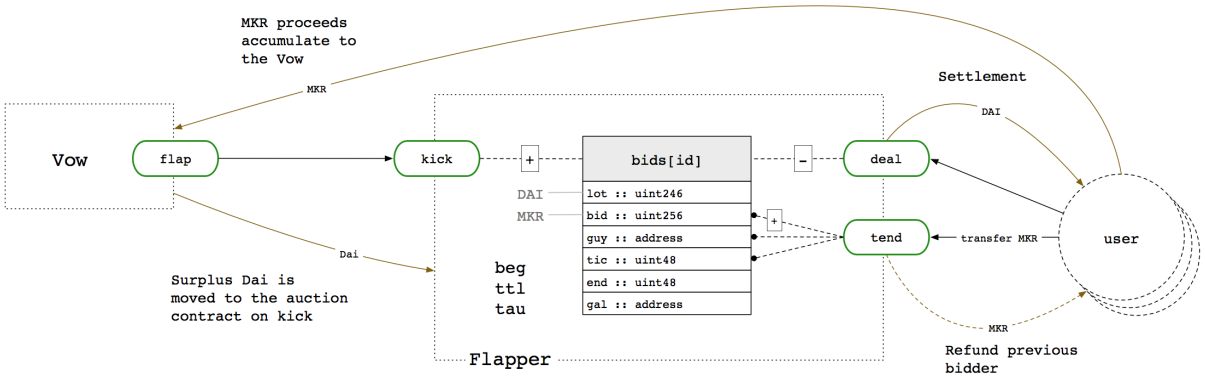
  合约地址: 0xA41B6EF151E06da0e34B009B86E828308986736D



尽快筹集Dai, 减少MKR通胀

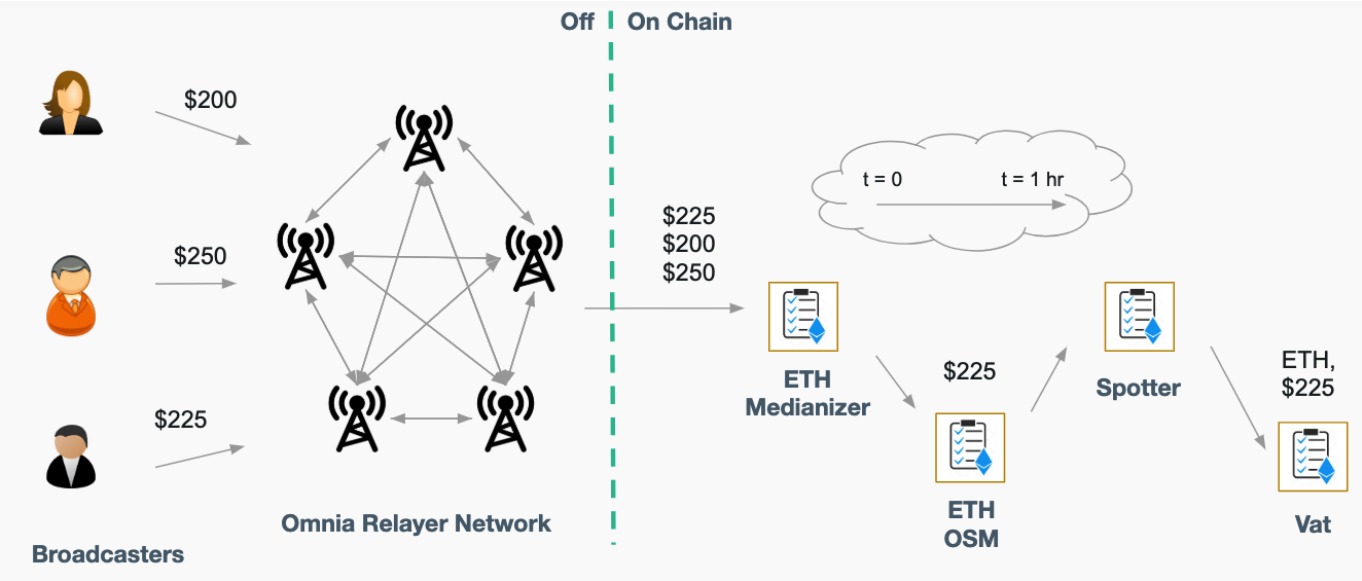
- Flap（盈余拍卖）用于消除Vow的盈余，通过拍卖固定数量的内部DAI来换取MKR。拍卖结束后，Flap会销毁中标者的MKR，并将内部DAI发送给中标者。

  合约地址: 0xC4269cC7acDEdC3794b221aA4D9205F564e27f0d



2.6 预言机模块

Maker协议预言机



- Maker协议中的预言机模块，它是用于向Vat合约提供不同抵押品类型的价格数据的模块。预言机模块引入了地址白名单的机制，允许白名单中的地址在链下广播价格更新，然后通过一个中值合约（Median）汇总，再通过一个安全模块（OSM）传递给Spot合约。Spot合约会从OSM中读取价格，并作为预言机和dss之间的联络人。

- 预言机模块有两个核心组件，分别是Median和OSM合约。

    - Median提供了Maker信任的参考价格。简单来说，它通过维护一个价格源合约的白名单，授权它们发布价格更新。每次收到一组新的价格时，计算它们的中值，并用它来更新存储的值。Median有权限控制的逻辑，可以让治理者添加和移除白名单中的价格源地址，并设置其他控制Median行为的参数，例如bar参数是接受新中值的最小价格数量。

    - OSM（Oracle Security Module）确保从预言机传播的新价格在被系统采用之前经过一定的延迟。OSM通过poke方法从一个指定的DSValue合约（或任何实现了read和peek接口的合约）读取值；read和peek方法会给出价格源合约的当前值，而其他合约必须被白名单才能调用这些方法。一个OSM合约只能从一个价格源合约读取，所以实际上每种抵押品类型都需要部署一个OSM合约。

  特点:

- 你可以直接从Median读取价格，这样你可以得到更实时的价格。但是这取决于更新的频率（poke调用的次数）。

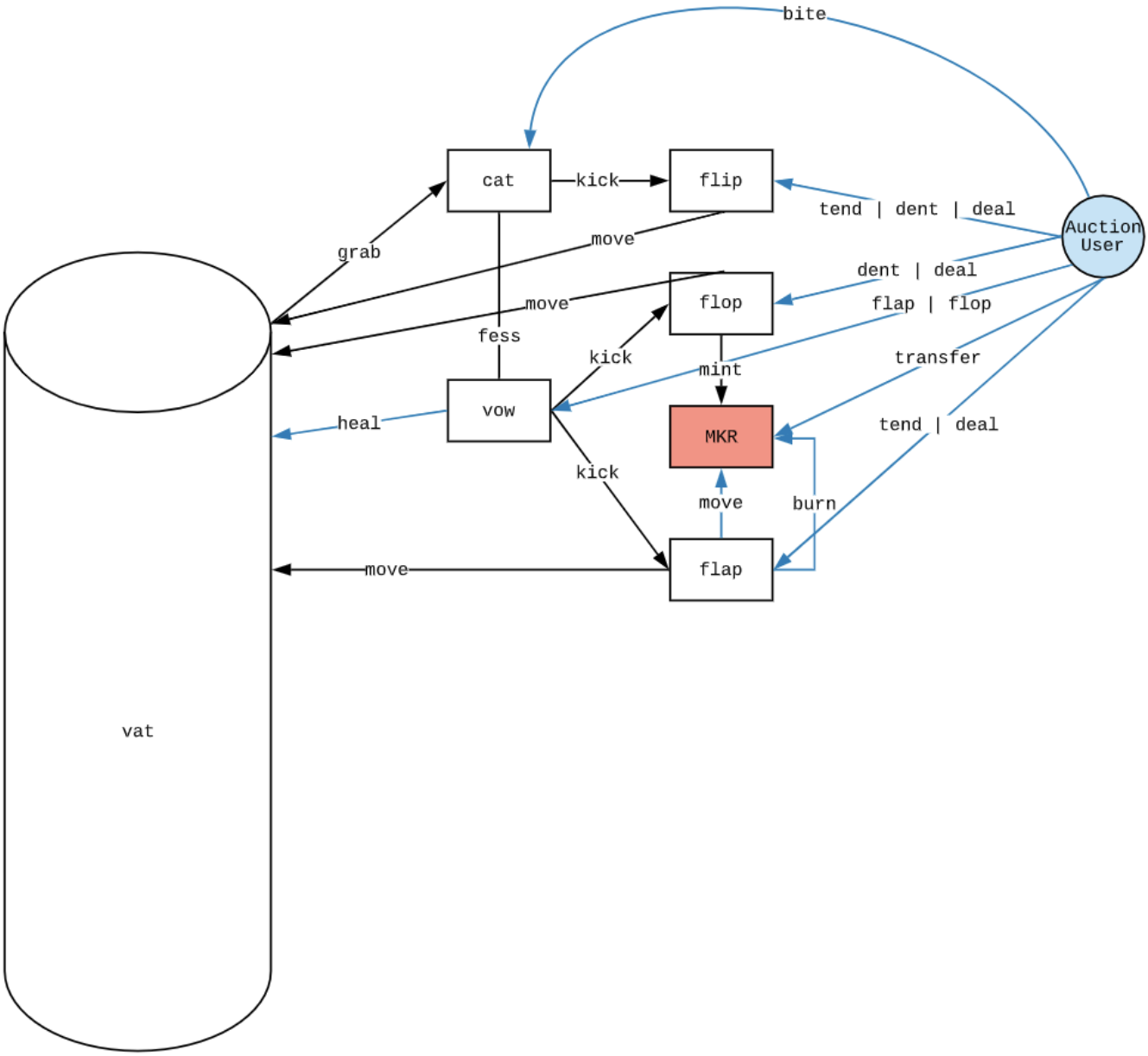
- OSM与Median类似，但有一个1小时的价格延迟。它有相同的读取过程（白名单、授权、read和peek），但工作方式不同。OSM不能直接更新，但可以poke它去读取一个具有相同结构（peek方法）的东西（在这里是Median，但你可以设置它读取任何符合相同接口的东西）。

- 每当OSM从一个源头读取值时，它会将读取到的值排队等待下一个小时或下一个跳跃属性（hop），该属性被设置为1小时（但可以是任何值）。当poke时，它会读取Median的值，并将其保存。然后之前的值变成了当前值，所以它总是落后于1小时。在一个小时过去后，当poke时，它保存的值变成当前值，而Median中的值变成未来值。
- spot - 如果你用一个ilk（例如ETH）poke它，它会从OSM读取，并且如果价格有效，就更新

2.7 MKR 模块

MKR是Make协议治理代币的实现.

合约地址: 0x9f8f72aa9304c8b593d555f12ef6589cc3a579a2



MKR代币是一个符合ERC20标准的代币，它除了提供标准的代币接口外，还有一些特殊的功能，如受DSAuth保护的销毁和铸造函数；通过MAX_UINT实现的二进制授权；以及用于transferFrom操作的push、pull和move别名。

MKR代币是Maker协议的核心组成部分，它有三种用途：

- 作为一种效用代币：当保险库（Vault）产生的Dai稳定费在协议中积累时，MKR持有者可以用MKR投票来启动Flapper拍卖合约，将Dai盈余出售给MKR。拍卖结束后，协议会销毁MKR。
- 作为一种治理代币：MKR被MKR持有者用来投票决定Maker协议的风险管理和业务逻辑。代币是投票权力的简单表示。
- 作为一种再资本化资源：MKR可以由Flopper拍卖合约自动铸造并出售给DAI，用于在协议出现资不抵债时进行再资本化。

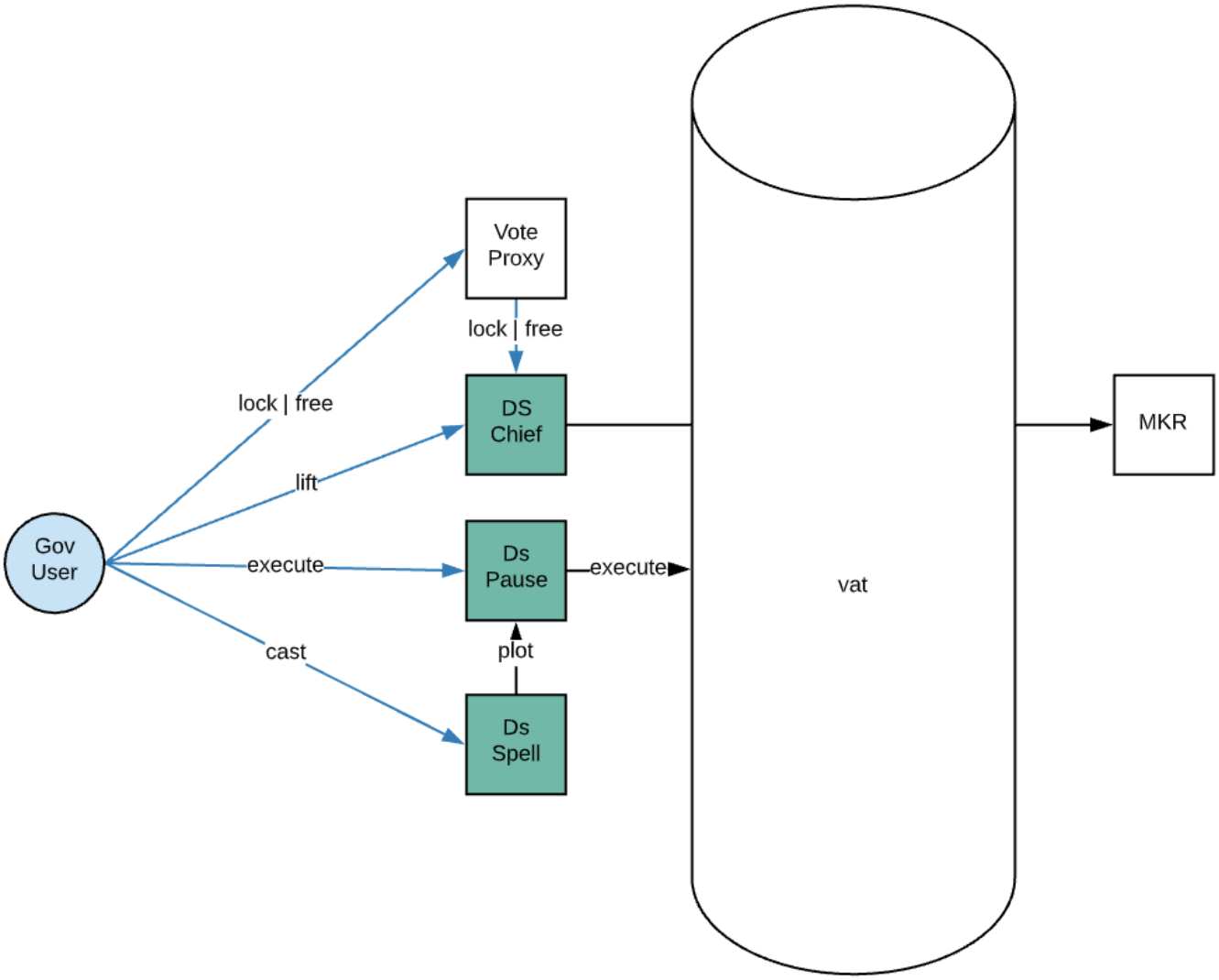
2.8 治理模块

Maker 协议治理合约

治理模块包含了一些合约，用于实现MKR投票、提案执行和投票安全等功能。

治理模块有三个核心组件，分别是Chief、Pause和Spell合约:

- Chief是一个通过赞成投票制度来选举一个“首席”合约的合约。这个首席合约可以与其他合约结合，例如DSAuthority，来为一个智能合约系统选定一套规则。
- Pause是一个基于delegatecall的代理合约，它有一个强制的延迟。这允许授权的用户安排一些函数调用，只有在预定的等待期过后才能执行。这个可配置的延迟属性设置了系统治理时使用的最小等待时间。
- Spell是一个没有所有者的对象，它只能执行一次一个动作或一系列原子性的动作（多个交易）。它可以被认为是一个没有所有者的一次性DSProxy（没有DSAuth混合，它不是一个DSThing）。



2.9 费率模块

Maker 协议的累积汇率机制

2.9.1 概述

核心机制是使用一个全局的累积率值（每种抵押品类型有一个），它可以乘以一个标准化的债务或存款金额，来得到总债务或总存款金额。这样可以避免每次累积时都要遍历所有的保险库和存款，从而节省气费和时间。

利率模块有两个主要组件，分别是Jug和Pot合约。

- Jug用于更新每种抵押品类型的累积率（rate），它由一个基础利率（base）和一个特定利率（duty）组成。Jug有一个drip函数，用于根据当前时间和利率计算新的累积率，并将其保存在Vat合约中。同时，Jug会将累积的稳定费发送给Vow合约，增加系统的总债务。

- Pot用于更新Dai储蓄率的累积率（chi），它由一个单一的利率（dsr）组成。Pot也有一个drip函数，用于根据当前时间和利率计算新的累积率，并将其保存在Pot合约中。同时，Pot会将累积的储蓄利息发送给Pot合约，增加系统的总债务，并从Vow合约中扣除相同的金额。

2.9.2 累积利率

累积利率R是一个随时间变化的值，它反映了稳定费用对Vault债务的影响。稳定费用是一种可变的利率，它根据MKR持有者的投票而定，目的是为了保持Dai代币的价格稳定在1美元左右。稳定费用在每秒钟都有一个值F_i，通常是一个接近1的数，比如1.000000000158153903837946258002097。这个数表示每秒钟债务增加的百分比。

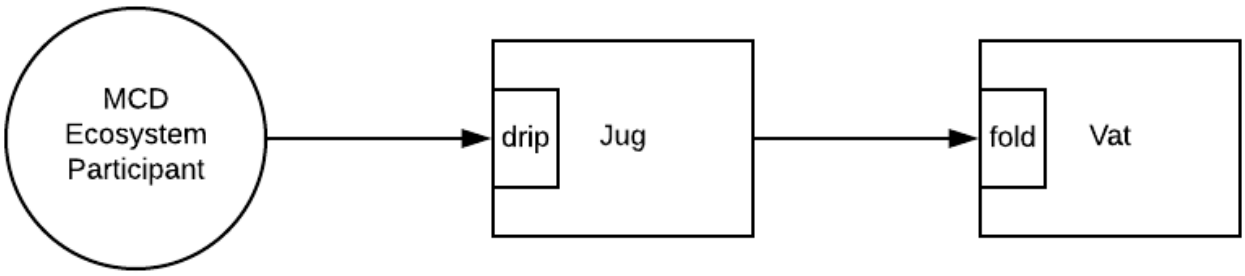
累积利率R在时间t_0时刻的初始值是R_0，假设为1。如果在时间t_0时刻，一个Vault创建并借出了D_0个Dai，那么它的归一化债务A就是D_0/R_0，也就是D_0。归一化债务A是系统存储在VaultDP中的一个参数，它表示在时间t_0时刻借出的Dai数量。

累积利率R在时间T时刻的值是由从t_0到T之间每秒钟的稳定费用F_i相乘得到的。也就是说，R(T) = R_0 * F_(t_0+1) * F_(t_0+2) * … * F_T。这个公式表示从t_0到T之间，债务每秒钟都按照F_i的比例增加。

Vault在时间T时刻的总债务D(T)就是归一化债务A乘以累积利率R(T)。也就是说，D(T) = A * R(T) = D_0 * R(T)。这个公式表示从t_0到T之间，债务增加了R(T)倍。

2.9.3 稳定费用累积

稳定费用累积是指在Vault（抵押债务仓库）中抵押资产并借出Dai时产生的一种可变利率费用。这个费用会随着MKR持有者对MakerDAO临时风险团队提出的提案的投票而变化。稳定费用的目的是为了平衡Dai代币的供需，因为在某些时期，Dai代币的市场价格可能会偏离其锚定的1美元。



稳定费用累积在MCD中主要涉及两个合约：Vat（系统的中央会计账本）和Jug（一个专门用于更新累积利率的模块），而Vow只是作为累积费用归属的地址。Vat合约记录了每个Vault中抵押资产和借出Dai的数量，以及每种资产类型对应的稳定费率。Jug合约负责根据MKR持有者投票结果更新每种资产类型的稳定费率，并计算其累积利率VaultDP所有者想要关闭或部分还Vault的CDP

时，他们必须支付相应的稳定费用，这个费用是根据他们借出Dai的数量和累积利率计算出来的。这个费用会以MKR代币的形式支付给Vow合约，从而减少MKR代币的总供应量。

Vat合约是系统的核心账本，它存储了每种抵押资产类型（Ilk）的累积利率（rate）和总归一化债务（Art）。Jug合约是一个专门用于更新累积利率的模块，它存储了每种抵押资产类型的每秒稳定费率，这个费率由一个全局基础值（base）和一个每种资产类型的特定值（duty）组成。给定一种抵押资产类型，它的每秒稳定费率就是它的duty和base的和。

当有人调用Jug.drip(bytes32 ilk)函数时，它会根据duty、base和上次调用drip的时间（rho）来计算ilk的rate的更新值。然后，Jug会调用Vat.fold(bytes32 ilk, address vow, int rate_change)函数，这个函数会：

- 把rate_change加到指定ilk的rate上
- 把Art*rate_change增加到Vow合约的盈余中
- 把Art*rate_change增加到系统的总债务（即发行的Dai）中

每个Vault（抵押债务仓库）都由Vat合约中的一个Urn结构体表示，它存储了一个“归一化债务”参数叫做art。任何时候代码需要VaultDP的总债务（包括稳定费用），都可以通过art*rate来计算（其中rate对应于相应的抵押资产类型）。因此，通过Jug.drip(bytes32 ilk)更新Ilk.rate实际上就相当于更新了所有用ilk代Vault的CDP的债务。

市场参与者有动机去调用drip函数，以及为什么他们会这么做。这些参与者包括：

- 寻求清算CDP的Keepers（因为稳定费用的累积会降低CDP的抵押率，使其进入不安全区域，从而让Keepers有机会清算它并从拍卖中获利）
- 想要借出Dai的CDP所有者（如果他们在借出Dai之前没有调用drip函数，他们会被收取从上次调用drip到现在的稳定费用，除非在他们还清CDP之前没有人调用drip，见下文）
- MKR持有者（他们有一个看到系统运行良好的既得利益，特别是收集盈余对于MKR代币的存亡是至关重要的）

尽管有各种激励因素，但由于燃气费用和公地悲剧效应，调用drip函数可能会不太频繁，直到达到一定的规模。

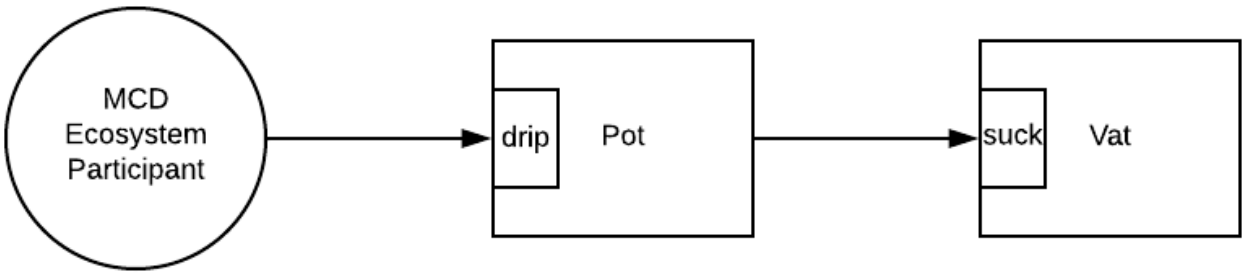
在两次调用drip函数之间，如果有债务的借出或还清，那么这部分债务不会被收取稳定费用。这是因为稳定费用是根据累积利率计算的，而累积利率只有在调用drip函数时才会更新。

根据稳定费用的更新时间和drip函数的调用时间，累积利率的实际值和理想值（如果每个区块都调用drip函数的话）可能会有一些差异。这可能会导致收取的稳定费用多于或少于理想值。

2.9.4 DSR 累积

Dai存款利率累积和稳定费用累积非常类似。它是通过Pot合约来实现的，它和Vat合约（以及Vow合约的地址）进行交互。Pot合约跟踪每个用户的归一化存款（pie[usr]）并维护一个累积利率参数（chi）。一个类似于Jug合约中的drip函数会被经济参与者间歇性地调用，以触发存款利息的累积。

Dai存款利率是由MKR持有者投票决定的，它允许用户把Dai存入Pot合约并获得一定比例的利息。这个利息通常会低于基础稳定费用，以保持系统的可持续性。



要点:

- 如果drip函数只是偶尔调用，那么chi的瞬时值可能会和理想值有所不同。这是因为chi是根据每秒的Dai存款利率计算的，而Dai存款利率只有在调用drip函数时才会更新。
- 代码要求在Dai存款利率变化之前必须调用drip函数，这样就可以消除由于Dai存款利率变化和drip函数调用不一致而导致的chi和理想值的偏差。
- chi是一个单调递增的值，除非有效的Dai存款利率变为负数（dsr < ONE）。这意味着Dai存款利率越高，chi增长得越快，用户从Pot合约中提取的Dai也越多。
- 系统没有存储用户从Pot合约中存入或取出Dai的记录。系统只存储了用户的归一化存款pie[usr]和累积利率chi。
- 系统也没有存储Dai存款利率dsr的变化记录。系统只存储了当前的dsr值和上次调用drip函数的时间rho。

如下市场参与者有动机调用Pot.drip:

- **任何从Pot合约中提取Dai的用户**（否则他们会亏钱！）因为如果他们不调用drip函数，他们就无法获得存款利息。
- **任何把Dai存入Pot合约的用户**——这不是经济上理性的，而是被智能合约逻辑强制要求的，因为在新的Dai被添加到Pot合约的同一个区块中，必须调用drip函数（否则，就会出现一个可以耗尽系统盈余的经济漏洞）
- **任何有动机增加系统债务的参与者**，例如希望触发flop（债务）拍卖的Keeper

如何设置dsr:

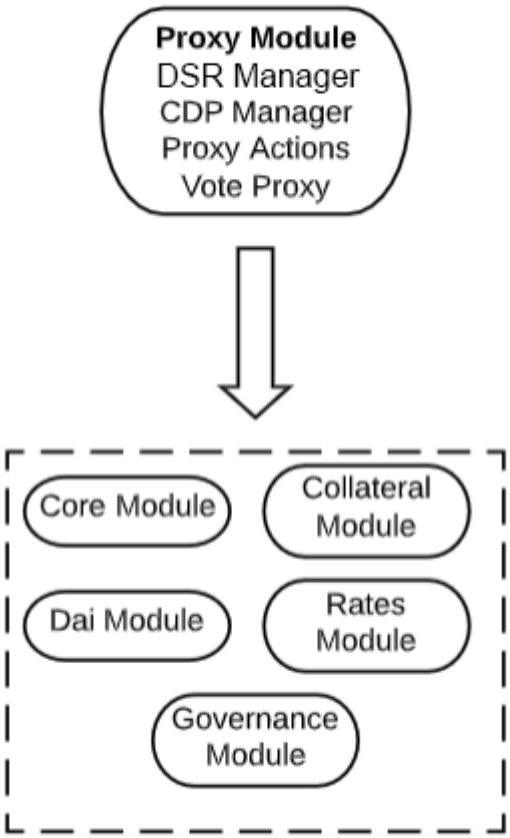
- Dai存款利率（DSR）是由MKR持有者投票决定的，它允许用户把Dai存入Pot合约并获得一定比例的利息。这个利息通常会低于基础稳定费用，以保持系统的可持续性。
- 如果想要设置DSR为每年0.5%，那么需要一个每秒的利率值r，使得在一年（N秒）内，r的N次方等于1.005。也就是说，r是1.005的N次方根。
- 为了计算r，可以使用任意精度计算器或者编程语言，把1.005除以N（31536000），然后取对数，再乘以10^27（因为DSR是用27位小数表示的）。得到的结果是：

  r = 1000000000158153903837946258002097
- 这个值就是要设置的DSR参数。它表示每秒借出Dai的数量增加了
  dsr=1000000000158153903837946258
- The `dsr` could then be set to 0.5% annually by calling:

  `Pot.file("dsr", 1000000000158153903837946258)`

2.10 代理模块

让用户更轻松的与Maker协议交互



代理模块是为了让用户或开发者更方便地与Maker协议进行交互而创建的。它包含了一些合约接口、代理和函数别名，这些都是用于DSR（Dai存款利率）和保险库管理以及Maker治理的必要功能。

代理模块有以下几个组成部分：

- **DSR管理器**：它提供了一个易用的智能合约，允许服务提供者把Dai存入或取出合约pot，以单个函数调用的方式开始赚取Dai存款利率，而不需要一个ds-proxy合约。这对于集成DSR功能的智能合约很有用。
- **CDP管理器**：它是为了实现保险库之间转移所有权的正式流程而创建的。简而言之，管理器通过一个dss包装器来让用户以一种简单的方式与他们的保险库进行交互，把它们当作非同质化代币（NFT）来处理。
- **投票代理**：它促进了在线投票和离线MKR存储。通过有一个投票代理，这让用户可以有一个关联的热钱包，可以从代理对应的冷钱包和DS-Chief（投票发生的地方）之间拉取和推送MKR。有两个主要原因要有/使用这个合约：
  - 支持两种不同的投票机制
  - 最小化MKR持有者需要把他们的钱包在线的时间
- **代理操作**：它被设计为由Ds-Proxy使用，Ds-Proxy是由用户个人拥有的，用于更容易地与Maker协议进行交互。注意它不是直接使用的（稍后会介绍）。代理操作合约被开发为用户ds-proxies的库。

2.11 闪电铸造(Flash Mint)模块

让任何用户都可以执行Dai的闪电铸造

合约地址: 0x60744434d6339a6B27d73d9Eda62b6F66a0a04FA

闪电铸造模块是一个允许任何用户执行Dai的闪电铸造的功能。闪电铸造是一种借用Dai并在同一笔交易中还清的操作，只需要支付一定的费用。这样，任何人都可以利用DeFi空间中的套利机会，而不需要提前投入资金。闪电铸造对Dai生态系统有很多好处，包括但不限于：

- 需要大量资金的漏洞会更快被发现，从而使DeFi空间更安全。
- 债务上限：任何单笔交易可以借用的Dai的最大数量。以rad单位编码为line参数。
- 铸造费用：在交易结束时必须返还给闪电铸造模块的额外Dai数量。这个费用会在成功铸造后转移到vow合约中。以wad单位编码为toll参数。

由于闪电铸造模块遵循了ERC3156标准，所以你可以直接使用该标准中的参考借款者实现：

如果用户想要在内部vat余额中移动Dai，而不是通过铸造/销毁ERC20 Dai来浪费燃气费，那么可以使用vat Dai闪电铸造函数来简化这个过程。

vat Dai版本的闪电铸造和ERC20 Dai版本大致相同，只有一些细微的差别：

```
vatDaiFlashLoan (IVatDaiFlashBorrower receiver, uint256 amount, bytes calldata
data)

flashLoan (IERC3156FlashBorrower receiver, address token, uint256 amount, bytes
calldata data)
```

注意，不需要指定代币，因为默认是vat Dai。另外，amount是以rad而不是wad为单位。

ERC3156规定使用代币授权来授权借款者向出借者还款。不幸的是，vat Dai没有一种方法来指定授权金额，所以我们选择让借款者在交易结束时推送欠款余额，而不是给闪电铸造模块完全的提款权限。

2.12 紧急关停模块

Maker协议的关闭机制

Maker协议是一个支持多抵押Dai的智能合约系统，它通过一种动态的组合方式，利用保险库、自治的智能合约和适当激励的外部参与者来支持和稳定Dai的价值。Dai的目标价格是1美元，相当于1:1美元的软锚定。紧急关闭是一种可以作为最后手段来直接保证Dai和保险库持有者获得目标价格，并保护Maker协议免受其基础设施攻击的过程。 紧急关闭会停止并优雅地结算Maker协议，同时确保所有用户，无论是Dai持有者还是保险库持有者，都能获得他们应得的净资产价值。简而言之，它允许Dai持有者在紧急关闭处理期后直接用Dai兑换抵押品。

2.12.1 概述

- 紧急关闭的启动过程是去中心化的，由MKR持有者控制，他们可以通过把MKR存入紧急关闭模块来触发它。
- 紧急关闭会在严重的紧急情况下启动，例如长期的市场非理性、黑客攻击或安全漏洞。
- 紧急关闭会停止并优雅地结算Maker协议，同时确保所有用户，无论是Dai持有者还是保险库持有者，都能获得他们应得的净资产价值。
- 保险库持有者可以在紧急关闭初始化后立即从他们的保险库中取回多余的抵押品。他们可以通过支持紧急关闭功能的保险库前端，例如Oasis Borrow，或者通过命令行工具来做这件事。
- Dai持有者可以在一个由MKR持有者决定的等待期后，用他们的Dai兑换系统中所有类型抵押品的相对份额。Maker基金会最初会提供一个网页来实现这个目的。
- Dai持有者总是从系统中获得相同比例的抵押品，无论他们是在第一批还是最后一批处理他们的索赔。

- Dai持有者也可以把他们的Dai卖给Keepers（如果有的话），以避免自己管理系统中不同类型的抵押品。
- 如果想了解更多关于Maker协议关闭的信息，可以阅读下面的End - Detailed Documentation以及紧急关闭模块文档。

2.12.2 紧急关停过程

2.12.2.1 概述

Maker协议是一个由智能合约、保险库和适当激励的外部参与者（如Keepers）组成的自动化系统，它可以调节Dai的供需。

Keepers是一些运行特定代码的实体，它们可以与Maker协议进行交互，执行一些有利于系统健康和Dai稳定的操作，例如参与拍卖、提供流动性或套利。Keepers对Maker协议非常重要，Maker基金会呼吁更多的人加入Keeper生态系统。

紧急关闭模块是一个智能合约，它有能力调用End.cage函数来启动紧急关闭。为了触发紧急关闭，需要50000个MKR存入紧急关闭模块。存入的MKR会立即被销毁。

紧急关闭有以下几个实现属性：

- **Dai无竞争条件**：每个Dai持有者都能兑换相同比例的抵押品，与他们交互合约的时间无关。
- **保险库优先**：保险库持有者可以在Dai持有者之前从他们的保险库中取回多余的抵押品。
- **负债务情况**：在紧急关闭时，可能存在个别保险库、整个抵押资产类型或整个系统的负债务情况，即债务价值超过了抵押品价值。因此，Dai持有者能兑换的抵押品价值可能会根据系统盈余或赤字而变化。因此，Dai持有者可能会收到少于或多于1美元价值的抵押品。

2.12.2.2 抵押物和Dai的兑换过程

- **Vault持有者的兑换过程**：保险库持有者可以在紧急关闭初始化后立即从他们的保险库中取回多余的抵押品。他们可以通过支持紧急关闭功能的保险库前端，例如Oasis Borrow，或者通过命令行工具来做这件事。
- **Dai持有者的兑换过程**：Dai持有者可以在一个由MKR持有者决定的等待期后，用他们的Dai兑换系统中所有类型抵押品的相对份额。这个份额是根据紧急关闭触发时的Maker预言机确定的抵押品价值来计算的。需要注意的是，Dai持有者无论何时处理他们的索赔，都能获得相同比例的抵押品。Maker基金会最初会提供一个网页来实现这个目的，以方便Dai持有者。
- **为什么紧急关闭优先考虑保险库持有者而不是Dai持有者**：三个主要原因：
  - **一致性**：在系统正常运行期间，过度抵押的保险库不会为系统中欠缺抵押的保险库补贴，所以紧急关闭也应该保持这种行为。唯一不同的是，潜在的损失会从MKR持有者转移到Dai持有者，因为无法预测紧急关闭后MKR代币的价值。
  - **激励**：给予保险库持有者优先权来取回他们多余的抵押品（如果他们的保险库没有欠缺抵押），可以激励他们维持过度抵押。这很重要，因为即使紧急关闭看起来很可能发

生，这种激励也依然存在，从而使系统更加强大。
    - **公平性**：紧急关闭时不会取消保险库持有者之前积累的稳定费用。保险库持有者可能会接受更高的费用，如果他们知道他们不会受到其他人的抵押水平影响，这可能会导致系统更高

2.12.2.3 拍卖结算

- **拍卖结算**：拍卖结算是指在紧急关闭期间，系统中的拍卖如何被处理的过程。拍卖是一种机制，用于清算过度抵押的保险库、出售系统盈余或回收系统债务。在紧急关闭期间，拍卖会被取消或冻结，以便让用户兑换他们的Dai和抵押品。
- **延迟期**：延迟期是指紧急关闭触发后，Dai和抵押品之间的兑换开始之前的一段时间。这个延迟期由MKR持有者决定，通常会足够长，以确保所有拍卖要么结束要么被跳过。但是，代码中没有保证这一点。
- **拍卖取消**：任何人都可以在任何时候取消抵押品（Flip）拍卖，这样就可以把抵押品收回到系统中，并让出价者取回他们的出价。而盈余（Flap）和债务（Flop）拍卖则会在调用Cage函数时被冻结。这两种拍卖都可以被调用来退还出价给最后一个出价者。

2.12.2.4 意图

- **紧急关闭的两种形式**：紧急关闭可能有两种主要形式。一种是触发紧急关闭，然后终止系统，没有未来重新部署的计划。这样，用户可以用多余的抵押品或用Dai兑换抵押品。
- 另一种是在有重新部署方案的情况下启动紧急关闭。这种情况可能发生在系统被触发进入紧急关闭事件，但是MKR持有者或第三方决定重新部署系统，并进行必要的改变来重新运行系统。这样，用户可以开启新的保险库，拥有新的Dai代币，同时从旧系统中兑换抵押品。

2.12.2.6 细节

很复杂, 9个步骤:

1. 锁定系统并关闭协议。
    - 锁定系统是通过冻结用户的各种操作，如创建金库、参与拍卖、存取DSR、投票治理等。
    - 关闭协议是通过停止和取消所有的债务/盈余拍卖，将资金分配给Dai持有者。这是因为在关闭期间，MKR的价值不确定，无法维持拍卖的正常运行。
    - 盈余拍卖被取消是因为盈余将被分配给Dai持有者。债务拍卖被取消是因为坏账将被分配给Dai持有者。
    - 抵押品拍卖不会立即取消，因为它们仍然与系统中的有价值的抵押品相关。抵押品拍卖可以继续运行，也可以被任何用户取消。
2. 设定协议中每种抵押品类型的最终价格。

- 最终价格是根据Maker预言机的价格数据来确定的。这是必要的，因为系统必须先处理系统状态，才能计算出最终的Dai/抵押品价格。特别地，我们需要确定两件事：
  - 考虑到抵押不足的金库，每种抵押品类型的缺口。
  - Dai发行的总量（总债务），即包括系统盈余/赤字后的未偿还Dai供应量。
- 首先，通过处理所有金库的一个函数来确定（a），该函数取消金库欠款的Dai（下文详述）。接下来，按照下文描述的方式确定（b）。

3. 取消金库中的欠款Dai，以确定系统债务

   - 取消欠款Dai是通过一个函数来实现的，该函数将多余的抵押品留在金库中，供所有者取回，然后取走用于偿还债务的抵押品。
   - 确定系统债务是通过处理拍卖中的Dai生成操作来实现的，以保证拍卖不会再产生更多的Dai收入。这样可以保证拍卖不会改变系统债务，包括抵押品拍卖不会再生成更多的Dai。
   - Dai生成来自于抵押品拍卖的第一阶段，因此如果所有拍卖都在第二阶段（出价减少的阶段），则说明生成已经结束。另外，DSR也必须在关闭期间关闭，以保证系统债务不变。
   - 例子：
     - 用户先出价越来越多的Dai，直到债务被覆盖。然后他们开始出价越来越少的抵押品。
     - 在第二阶段的拍卖不再影响系统债务，因为Dai已经被收回。最后，对于在第一阶段的拍卖，可以被取消，并将抵押品和债务返回到金库中。
     - 有两种方法可以保证抵押品和债务返回到金库中：
       - 处理冷却时间（债务队列的长度）；或
       - 取消进行中的拍卖。

4. 即使用冷却时间或取消进行中的拍卖。这一步的目的是确保拍卖不会再产生更多的Dai，从而影响系统债务的计算。这一步有两种方法：

   - 使用冷却时间。冷却时间是一个预先设定的时间段，用于取消欠款Dai和进行中的第一阶段拍卖。冷却时间只需要足够长，以处理所有抵押不足的金库和拍卖。实际上，它可以很短（例如5分钟）。但是，由于可能发生网络拥堵等情况，它可能会设定得更长。
   - 取消进行中的拍卖。取消进行中的拍卖是通过一个函数来实现的，该函数将取消所有进行中的拍卖，并收回抵押品。这种方法可以更快地处理拍卖，但需要更多的处理调用。这种方法可以让Dai持有者更快地取回他们的抵押品。下一步是取消每个进行中的第一阶段抵押品（Flip）拍卖，并取回所有的抵押品和Dai。然后，第二阶段（反向）拍卖可以继续正常运行，通过设置冷却时间或取消进行中的拍卖。 注意，这两种方法在这个实现中都是可用的，而且可以对每个拍卖单独启用取消进行中的拍卖。当一个金库被处理并且没有剩余债务时，剩余的抵押品可以被移除。

5. 第五步是从已结算的金库中移除剩余的抵押品。在金库被按照最终价格结算，并且欠款Dai被取消后，金库所有者可以按需调用这个过程。它将移除第三步后剩余的所有抵押品，也就是没有用于偿还债务的那部分抵押品。如果用户在结束时没有债务在金库中，他可以跳过第三步和第四步，直接进行这一步来释放他的抵押品。

6. 第六步是稳定Dai的总发行量。在处理期结束后，可以使用thaw函数来确定每种抵押品类型的最终价格。这里假设所有抵押不足的金库都被处理了，所有拍卖都已经结束了。这个函数的目的是稳定Dai的总发行量。注意，在这个阶段，需要检查核心金库引擎中的Dai盈余是否为零。这个要求是保证系统盈余已经被考虑在内的。此外，这意味着在你可以稳定Dai的总发行量之前，你必须取消尽可能多的金库中的欠款Dai，以取消Vow中的任何Dai盈余。取消Dai盈余是通过在你可以稳定Dai的总发行量之前，取消系统资产负债表中的盈余和债务来实现的。

7. 第七步是确定了给定抵押品类型的交换价格的计算，以及在出现赤字/盈余的情况下可能对最终交换价格的调整。在此机制的这一点上，已经计算出每种抵押品类型的最终价格；Dai持有者现在可以将他们的Dai转换为抵押品。每个Dai单位都可以兑换一定数量的抵押品。Dai持有者必须先锁定他们的Dai，以便准备将其兑换为抵押品。一旦Dai被锁定，它就无法解锁且不可转移。稍后也可以锁定更多的Dai。

8. 第八步将抵押品分配给已经锁定Dai以进行兑换的Dai持有者。锁定的Dai数量越多，可以释放给Dai持有者的抵押品就越多。

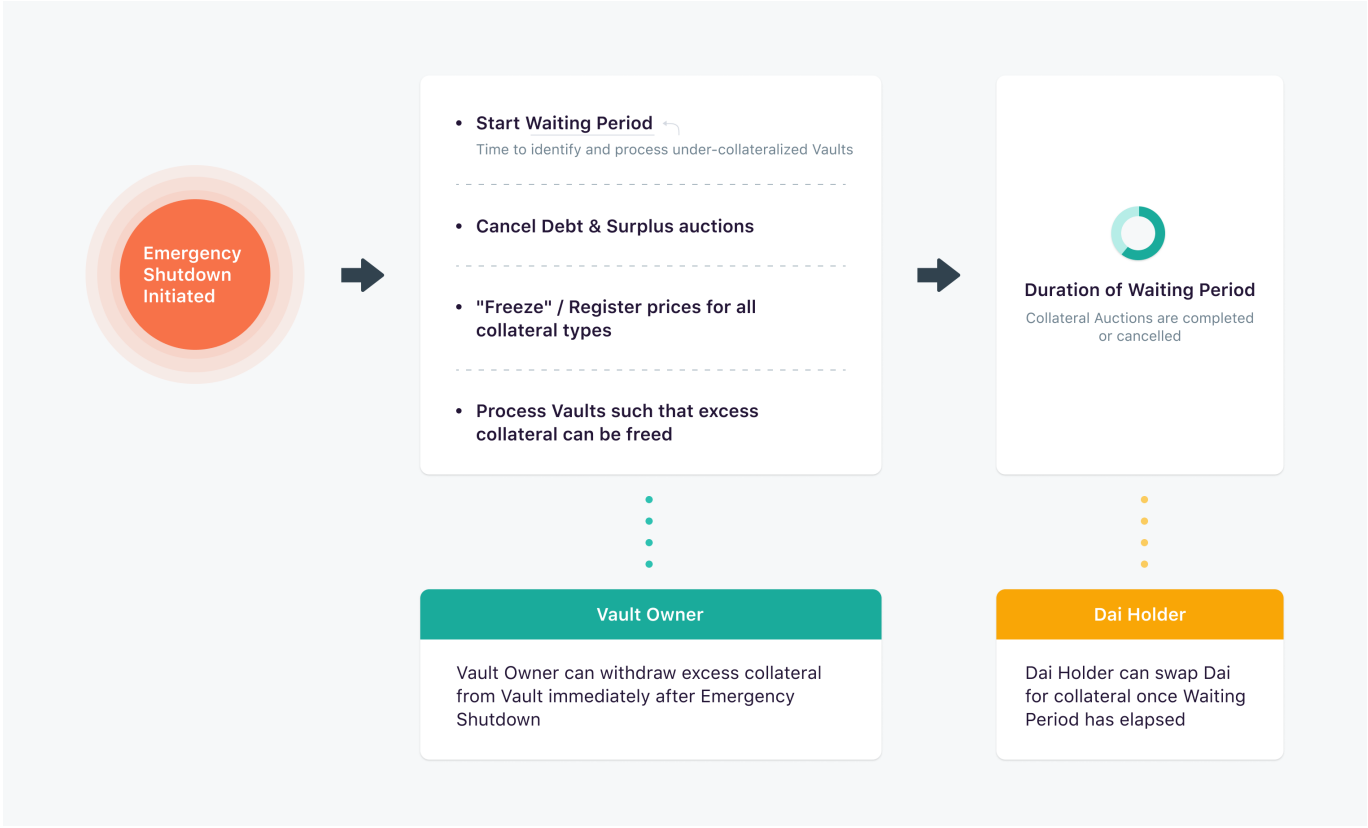
9. 第九步将锁定的Dai兑换为抵押品。请注意，抵押品代币的数量将受用户锁定Dai数量的限制。这意味着，用户锁定的Dai越多，可以释放给Dai持有者的抵押品就越多。

2.12.3 End 合约

关停系统

合约地址: 0xaB14d3CE3F733CACB76eC2AbE7d2fcb00c99F3d5

End合约是负责协调系统的关闭（Shutdown）过程。Shutdown是一种在系统需要升级或遇到安全问题时，关闭系统并退还Dai持有者相应的抵押品的过程。



Cage 是协议中最复杂的机制. 只是因为Cage必须改变系统中几乎所有组件的行为，并且能够在各种可能的抵押品不足的情况下运行。下面列出了一些关键的属性，比如Dai和Vault的对等性，或者缺乏竞争条件，这些都是Shutdown的理想（可有可无）属性，但实际上并不是Shutdown的真实实现所满足的。

- Dai对等性 - 假设系统中有足够的抵押品，那么从1 Dai兑换的每种抵押品的价值之和等于目标价格（即1美元），以Cage使用的抵押品价格为准。
- Vault对等性 - 每个Vault都按照全局结算时的抵押品价格结算。例如，在Cage之后，每个Vault中剩余的抵押品的价值等于Cage之前的Vault的净值，以Cage使用的抵押品价格为准，或者零，取较大者。
- Dai无竞争条件 - 无论Dai持有者何时与合约交互，都能兑换相同数量的每种类型的抵押品。这是最重要的属性，因为它保证了所有Dai持有者的公平性。
- 近乎即时的Dai兑换 - 在Cage之后，所有Dai都可以立即兑换为抵押品。
- 近乎即时的Vault兑换 - 在Cage之后，所有剩余的抵押品都可以立即取回。
- 无需链下计算 - 系统不需要Cage授权方提供任何链下计算的值。例如，它可以完全依赖最后一个OSM价格提供值。

当前实现的shutdown:

- Dai无竞争条件 - 每个Dai持有者都能按照相同的比例兑换不同类型的抵押品，无论他们何时与合约交互。这保证了所有Dai持有者的公平性。

- Vault优先 - Vault持有者被优先考虑，他们可以在Dai持有者之前取回他们多余的抵押品。如果Shutdown时，有些Vault或者整个系统是低于100%抵押率的，那么Dai持有者会受到“剪发”的影响，也就是说，他们兑换的抵押品价值可能低于1美元。
- 立即Vault兑换 - 在Shutdown开始后，Vault持有者可以立即取回他们的抵押品，只要他们一次性执行所有合约调用。
- 无需链下计算 - 系统不需要Shutdown授权方提供任何链下计算的值（例如，它可以完全依赖最后一个OSM价格提供值）。
- Vow缓冲区帮助 - 在Shutdown开始后，Vow中的任何盈余（和坏账）会按比例分配给所有Dai持有者作为奖励（或惩罚）。例如，如果系统总债务中有10%是Vow中的净盈余，那么Dai持有者会多获得10%的抵押品。

Shutdown机制的九个步骤是：

- cage() - 锁定系统并启动Shutdown。这是通过冻结用户操作，取消flap和flop拍卖，锁定系统其他合约，禁用可能干扰结算过程的某些治理操作，以及开始冷却期来实现的。
- cage(ilk) - 设置每种抵押品的最终价格（tag）。这是通过读取价格预言机来实现的。这是必需的，因为我们必须先处理系统状态，才能计算出最终的Dai/抵押品价格。
- skim(ilk, urn) - 按照tag价格结算Vault / 取消Vault欠的Dai。多余的抵押品仍留在Vault中供所有者取回。然后，取走支撑债务的抵押品。
- wait或skip - wait设置冷却期。冷却期只需要足够长，以确保所有拍卖都结束或被跳过，但代码中没有保证这一点。skip可以取消进行中的拍卖并没收抵押品。这允许更快地处理拍卖，但需要更多的调用。
- free(ilk) - 从已结算的Vault中移除（剩余）抵押品。这只有在Vault没有债务时才能发生。
- thaw() - 在所有skim之后修正Dai供应 / 修正稳定币总供应量。注意，它可能还需要额外处理Vault以弥补vow盈余。
- flow(ilk) - 计算每种抵押品的固定价格（fix），可能根据盈余/赤字调整tag价格。
- pack(wad) - 锁定Dai以备cash / 将一些稳定币放入bag中，准备兑换抵押品。
- cash(ilk, wad) - 用pack好的Dai兑换抵押品 / 用bag中的一些Dai兑换给定类型的gem，按照bag大小分享比例。

2.12.4 ESM 合约

Maker协议关停的触发系统

合约地址: 0x09e05fF6142F2f9de8B6B65855A1d56B6cfE4c58

ESM可以由MKR持有者通过投票激活，他们可以向ESM存入MKR来启动Shutdown。MKR是Maker Protocol的治理代币，用于参与系统参数的调整和重要决策的投票。当ESM内部的Sum变

量达到或超过最小阈值（min）时，任何人都可以调用ESM的fire()和denyProxy()方法。fire()方法会调用End.cage()，开始Shutdown过程。

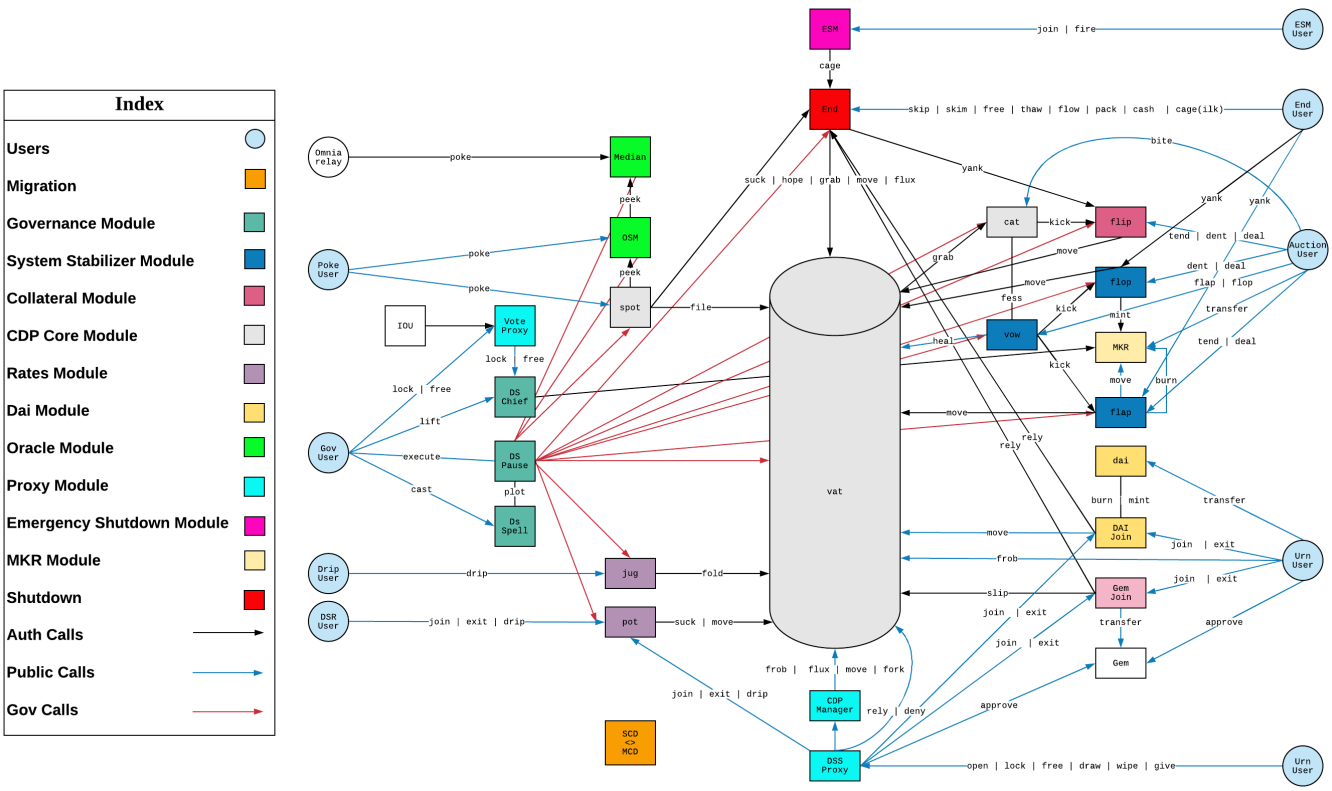
ESM可能在以下几种情况下使用：

- 防止恶意治理
- 防止严重漏洞的利用（例如允许窃取抵押品的漏洞）
- 在恶意治理攻击的情况下，存入ESM的MKR持有者不会期望恢复他们的资金（因为这需要恶意多数通过必要的投票），他们唯一的选择是建立一个替代的分叉，在其中削减多数方的资金并恢复他们自己的资金。
- 在其他情况下，剩余的MKR持有者可能会选择通过铸造新代币来退还ESM存入者。

注意：治理可以通过调用cage()（这与End.cage()不同）来解除ESM。

3. 分析

下面这幅图包含了Maker协议所有的模块, 以及模块之间的关系. 根据上面的模块以及合约的说明, 我们可以轻易的找出每个功能调用路径是什么.



4. 部署

dss 合约可以使用dss-deploy这个合约部署: