**AAVE协议白皮书V2.0**

[wow@aave.com](mailto:wow@aave.com)（中文译者：林科）

2020年12月

**摘要**

本文档描述了Aave协议V2背后的定义和理论，包括它如何改进V1、新特性和实现的不同方面。

# 1 介绍

Aave协议标志着从分散的P2P借贷策略(贷款人和借款人之间的直接贷款关系，如ETHLend)向基于池的策略的转变。贷款不需要单独匹配，相反，它们依赖于池资金、借款金额及其抵押品。这使得即时贷款具有基于池的状态的特征。

下面的文章描述了Aave协议V2不同于V1的关键功能。开发Aave V2的主要驱动力是改进在V1中实现的次最优解决方案（译者注：V1的方案不够完美，需优化）

* Inability to upgrade the aTokens（无法升级aTokens）
* Gas inefficiency（gas效率低）
* Architecture simplification favoring automated testing through fuzzers and formal verification tools（架构简化有利于通过fuzzers和形式化验证工具进行自动化测试）
* Code simplification（代码简化）

目前在V1中实现的一些解决方案是在以太坊网络明显不同的时候设计和开发的。DeFi的指数级增长使交易数量增长了两倍，而Istanbul版本带来的gas成本上涨进一步给交易带来了压力。

此外，Aave V2还提供了以下新特性

* 债务代币化（通证化）。借方的债务现在由代币表示，而不是用合约的内部accounting表示，从而使：
  + 关于协议的借款方面的代码简化 - 一些计算现在隐含在债务代币的铸造/燃烧逻辑中。
  + 具有可变的和多重稳定利率的同时借款。在V1模式中，借款人的贷款要么是可变的，要么是稳定的，以前的贷款自动转换为最新的贷款利率。在V2中，用户可以结合可变和多个稳定利率头寸持有多笔贷款。多个稳定利率头寸通过采用稳定利率贷款的利率的加权平均来处理。
  + Native credit delegation（原生信贷委托）- 通过委托的概念，用户可以通过开通信贷额度来委托给其他地址。可以在上面开发一些有趣的工具，例如：
    - 从一个冷钱包里借钱 - 用户可以把他们的抵押品放在冷钱包里，仅仅把他们的信用委托给一个热钱包去借钱。整个头寸可以从热钱包管理，比如借款、偿还和添加更多抵押品;而资金则被放在一个安全的冷钱包里。
    - Credit delegation facilities（信贷委托设施） - 只要用户接收到其他提供抵押品的用户的委托，用户就可以低抵押贷款。
    - Automated yield farming（自动收益耕作，流动性挖矿） - 信贷委托使构建工具，以自动开放信贷额度的在多种协议上的耕作收益（流动性挖矿）成为可能。
* 闪电贷V2。闪电贷是Aave V1的颠覆性功能，它允许创建各种工具来进行再融资、抵押品交换、套利和清算。闪电贷已成为一种强大的DeFi机制。在最初的V1闪电贷中，一个重要的限制是它不能在Aave中使用。Aave V2实现了一个解决方案，使闪电点与Aave协议的任何其他功能结合使用，为许多新的可能性提供了支持。
  + 抵押品交易。在不需要结清债务头寸的情况下，将一种/多种抵押品转换为另一种。
  + 用抵押品偿还一笔贷款。使用协议中的抵押存款来偿还一笔债务。
  + Margin trading保证金交易（交存一定保证金不支付全部货款的交易）。创建可用于以后交易的保证金头寸。
  + 债务交换。将债务敞口从一种资产转换为另一种资产。
  + 保证金存款

# 2 架构说明书

图示

描述已自动生成

图1:Aave V2协议架构

图示

描述已自动生成

图2:Aave V1协议架构

一些主要的架构差异将图1中的V2与图2中的V1的结构区分开来。

1. 以前存储在LendingPoolCore合约中的资金现在存储在每个特定的aToken中。这使得协议能够更好地在资产之间进行隔离，这有利于实现可用于收益耕作的aTokens。
2. LendingPoolCore和LendingPoolDataProvider已被删除并替换为库。这在优化代码复杂性和冗长性的同时，将所有操作的gas消耗减少了15 - 20%。
3. 现在所有的操作都通过LendingPool进行;替换之前每种aToken需要的redeem()。
4. 债务代币跟踪用户的债务。

# 3 特性说明

## 3.1 aTokens

V2 aTokens带来以下更新

* 支持EIP-2612
* 在基础aToken的实现中没有利率重定向 - 这可能会在未来重新引入

以下V1中的定义在V2中依然成立:

* 当前流动比率

总体借款率和使用率的函数，

* ，累积流动性指数。

reserve（储备金）在时间间隔∆T内累积的利息，每当发生借款、存款、偿还、赎回、交换、清算事件时更新。

文本

描述已自动生成

* ，reserve正常的收入。

由reserve累积的持续利息，

在V1中，每次操作后，被存储为用户x指数NI ( x )，对于一个用户x，累积的aToken余额计算如下:



在V2中，用户指数实际上作为一个storage变量消失了，并与主余额以ratio（比例）的形式一起存储，称为Scaled Balance, ScB（译者注：用户资产占池子总资产的比例）。每一个操作导致用户余额的计算，导致余额增加或减少，即铸造或燃烧aTokens。

* 存款。当用户在协议中存入金额m时，他的Scaled Balance更新如下:

文本

中度可信度描述已自动生成

* 取款。当用户在协议中提取金额m时，他的Scaled Balance更新如下:



在任何时间点，一个用户的aToken余额是:



## 3.2 债务通证化

包括每秒累积的债务在内的债务代币的总供应量定义如下



一项资产在t时刻的总负债定义为:



这个定义取代了V1白皮书1.2节中描述的总体稳定或可变借款。

## 3.3 可变债务

V1白皮书介绍了以下适用于V2的定义:

* ，累计可变借款指数。

在时间T期间，可变借款V B以可变利率V R累积的利息，每当发生借款、存款、偿还、赎回、交换、清算事件时更新。



* V I(x),用户累计可变借款指数。

特定用户的可变借款指数，在用户打开可变借款头寸时存储。



* PB(x), 用户主借款余额

当一个用户打开一个借款头寸时存储的余额。在multiple borrows（多重借款，译者注：不知道这样翻译是否准确）的情况下，复利每次累积，它成为新的主借款余额。

在V2中，债务代币遵循与V1中的aTokens相同的不断增长的逻辑。可变债务代币遵循scaled balance方法。引入了标准化可变(累计)债务的概念



为用户x在t时刻的scaled balance，m为交易金额，为标准化可变债务:

* 借款。当用户x从协议借款m时，scaled balance将更新



* 偿还/清算。当用户x偿还或被清算的金额为m，scaled balance更新



在任意时刻，一个用户的总可变债务余额可以写成:



这个可变债务令牌余额替代了V1白皮书的1.2节中描述的用户变量借款余额。

## 3.4 稳定债务

对于稳定利率的债务，以下V1定义依然成立:

* ，总体稳定利率
  + 当一笔数额为的稳定借款以的利率发行时:



当用户x以稳定利率偿还稳定借款i，数额为时:

文本

描述已自动生成

存储在每个特定货币的稳定利率代币中。用户x的稳定债务代币SD (x)余额定义如下:



在V1中，用户x的稳定利率SR (x)总是等于上一笔贷款的稳定利率，之前的贷款在新贷款的基础上进行再平衡。从V2开始，计算经过了i笔稳定贷款后每项资产储备的稳定利率SR (x):



## 3.5 闪电贷V2

流程图图3描述了V1 Flash Loans。检查资金是否已转出的步骤如下:

1. 在将资金转移到执行者U之前，协议对闪电贷资产的余额进行快照
2. 在闪电贷结束时，一个检查强制执行合约的余额必须包括闪电贷的借款金额加上闪电贷的premium（保费，额外费用）。如果金额不正确，交易将被回滚。

这限制了在Aave内部使用Flash Loans，因为允许其他行为(尤其是存款、还款和清算)可能会使协议面临重入攻击的风险。因此，这些操作通过可重入保护机制进行互斥。

图示

描述已自动生成

图3:Aave V1闪电贷

V2提供了一种新颖的防止重入攻击的保护，允许在所有Aave交易中使用闪电贷。

解决方案

最初的实现选择了特定的ETH代码路径;不幸的是，ETH不支持pull支付策略。在这个新版本2中，所有ETH特定的代码都被删除了，产生了以下变化:

1. 闪电贷是从资金转移到闪电贷执行者u开始的
2. 最后，资金从闪电贷执行者u以等于借款资金加上费用的金额转回来(pulled back)。如果拉取资金不成功，闪电贷失败;例如，当资金缺乏approve或缺乏资金来偿还债务时。

这从技术上消除了对前后余额进行快照的需要，因此消除了对重入检查的需要。

此外，闪电贷现在支持多种模式:闪电贷调用者可以决定保持贷款开放，须有适当的流动性来满足抵押品要求。

## 3.6 抵押品交易

Aave协议V2提供了一种交换存款资产的方法，也适用于抵押品。这是通过以下方式利用v2闪电贷实现的:

* flashLoan()函数由用户调用，把一个接收方合约的地址作为参数传进去，这个接收方合约实现了IFlashLoanReceiver接口，该合约是一个不可信的合约，需要由用户提前验证;要交换的基础资产的列表、这些资产的数量的列表、一个包含要交换的资产的额外参数和用户选择的最大滑点，都是编码的。
* 然后，接收方合约将使用收到的资金将其交换到目标资产，代表用户再次存款，并提取用户在闪电贷货币中的存款，以偿还闪电贷。

一个例子如下：

1. 一个用户在协议中存储了100个LINK和20个UNI（译者注：作为借贷的抵押品），并有100 USDC的债务。他想把他在协议中的存款LINK和UNI都换成AAVE，而不需要在一次交易中偿还任何债务。
2. 用户调用flashLoan()函数，将接收方合约的地址作为参数传递，接收方合约包含执行操作的逻辑，以及LINK和UNI的地址;100和20作为金额，需要编码，要交换的资产（AAVE）的地址，2%作为交易的最大滑点。
3. 接收方合约将使用去中心化交易所（译者注：比如uniswap）将指定数量的LINK和UNI换为AAVE。
4. 接收方合约将代表用户在Aave协议中存入交换得来的AAVE。
5. 接收方合约将从用户那里转移aLINK和aUNI中等值的闪电金额，将其赎回到LINK和UNI，并将批准pool在闪电贷结束时提取这些资金以偿还。

## 3.7 用抵押品偿还借款

该特性也使用Flash Loans v2实现，该特性允许用户将存储在协议中作为抵押品的一个或多个资产，用来偿还部分或全部债务头寸。与Swap Liquidity一样，通常所有的特性都基于Flash Loans，用抵押品来偿还债务使用flashLoan()函数和一个接收方合约，这个合约实现了IFlashLoanReceiver接口。这个接收方合约被传入以下参数：用于flash、交换和偿还的抵押资产列表,这些抵押资产的金额的列表,需要编码,需要偿还债务的资产的列表,需要偿还债务的资产的金额的列表,每种债务资产的借贷模式(稳定或可变)的列表。需要注意的是，在这种情况下，接收方合约将期望收到需要偿还多少的确切金额，与Swap Liquidity不同，在Swap Liquidity中，交换的预期金额是确切的抵押品金额。

这个特性的流程是

1. 一个用户在协议中存放了100个AAVE，并以可变利率借了200个USDC。由于他目前没有可用的USDC资金来偿还贷款，他想将他的AAVE抵押品的一部分换成USDC，并用它来偿还贷款。
2. 用户调用flashLoan()函数，传递如下参数:接收方合约的地址，该合约包含操作的逻辑,AAVE的地址,7作为抵押品交换数量(估计AAVE可以偿还200 USDC债务),200作为偿还的债务额，借贷模式为可变。
3. 接收方合约将使用去中心化交易所将7个AAVE换为USDC。
4. 接收方合约将使用交换产生的USDC来代表用户偿还协议中的USDC债务。
5. 一旦债务得到偿还，接收方合约将从用户那里转移返回闪电AAVE所需的aAAVE金额，将其赎回为AAVE，并将批准pool在闪电贷交易结束时提取这些资金。
6. 如果在swap结束时7 AAVE有任何剩余，这些资金将代表用户存放在协议中或直接发送到他的钱包。

## 3.8 信用委托

除了债务通证化之外，V2还支持信用委托:borrow()函数支持到不同地址的信用额度，只要调用者地址被授予了信用额度，就不需要抵押品。

该功能通过每种债务通证上的approveDelegation()函数实现。用户将能够为特定的债务模式(稳定或可变)提取他们的额度。borrow()函数有一个onBehalfOf参数，用于调用者指定用于提取信用的地址（译者注：即谁授予了调用者，委托方）。

信用委托的实现需要一些权衡取舍:

* 一个委托方可以将信用委托给多个实体，但一个被委托方只能一次从一个委托方获得信用。被委托方不能将所有委托方的债务聚合在一个borrow()中
* 委托方可以同时将稳定信用和可变信用委托给实体，但被委托方不能从一次borrow()中同时获得可变信用和稳定信用

# 4 Gas优化和改进

## 4.1 pow函数的实现

在最初的版本中，复利的计算依赖于Babylonian方法实现的指数公式，导致更多的时间消耗和昂贵的执行。

V2版本通过将这个指数公式近似为一个二项式展开来优化执行成本，这在一个较小的基数下工作得很好。实现使用以下二项式展开



函数calculateCompoundedInterest, (MathUtils.sol line 46)实现了前三个扩展，这为5年的贷款期限提供了一个很好的复利近似值。这导致了一个轻微的支付不足，但gas优化带来的好处抵消了这个不足。

需要注意的是，对于可变和稳定借款，则略有不同：

* 对于可变借款，任何借款人的任何操作都应计息
* 对于稳定的借款，只有当特定的借款人执行一项操作时，才会产生利息，这增加了近似的影响。不过，考虑到交易成本的节省（译者注：gas费的节省），这种差异似乎是合理的。

## 4.2 从WadRayMath和PercentageMath移除SafeMath

V1 WadRayMath库在内部使用SafeMath来保证操作的完整性。经过深入的分析，SafeMath在协议的关键领域产生了密集的高成本，每次调用消耗30 gas fee。这支持WadRayMath的重构来移除SafeMath，这在某些操作中节省了10-15k的gas。

## 4.3 用一个bitmask映射用户的贷款/存款

在最初的V1版本中，协议循环遍历所有活动的资产，以识别用户的存款和贷款。这将导致高gas消耗和降低可扩展性 - 因为随着协议中的资产越多，取款/借款/偿还/清算资产的成本将增加。考虑两个改进的想法

1. 对于每个用户，保持一个被用作抵押品/借款的资产列表，当用户存款/取款/借款/偿还时更新。在计算总系统抵押品时，函数calculateUserAccountData()可以针对特定用户的资产，而不是遍历所有被协议支持的资产。

这个解决方案没有任何限制，尽管它更加gas密集型 - 当考虑从这个列表中删除项目的成本，检查每个aToken的余额，稳定或可变债务，不管用户是存款，借款还是两者都有。。。

形状

描述已自动生成

图4:用户抵押品/借款Bitmask结构

1. 用结构图4创建一个位掩码。位掩码的大小为256 bit，它被分成一对一对的bit（译者注：2个bit），每个bit对对应一种资产。bit对中的第一个bit表示用户是否将一种资产用作抵押品，第二个bit表示用户是否借用了某种资产。这个实现强加了约束：
   * 只能支持128种资产，为了增加更多的资产，需要使用另一个uint256。
   * 对于帐户数据的计算，协议仍然需要查询所有列出的资产。

与基于列表的解决方案相比，它具有以下优点

* + 额外的资产只需要5k gas，而不是基于列表方案的20k gas
  + 极其便宜的验证用户资产的借款(0xAAAAA...!=0)或存款(configuration=0)
  + 通过获取aTokens/debt代币余额，立即访问哪些资产正在存入/借出/两者兼有
  + 在calculateUserAccountData()中，配置可以在开始时缓存，并用于执行所有计算，这在SLOADs时大大节省了gas。

这两种解决方案都得到了实现和测试，由于128种资产的限制提供了增长空间，第二种解决方案是首选的，考虑到Aave的多种市场(multimarkets)，这种增长空间更大。

## 4.4 用一个bitmask进行储备池(reserve)配置

还引入了一个位掩码来存储图5中定义的储备池配置。

通过使用uint32和布尔值也可以实现类似的打包，位掩码从更高的gas效率中受益，当一次更新多个配置时更是如此。

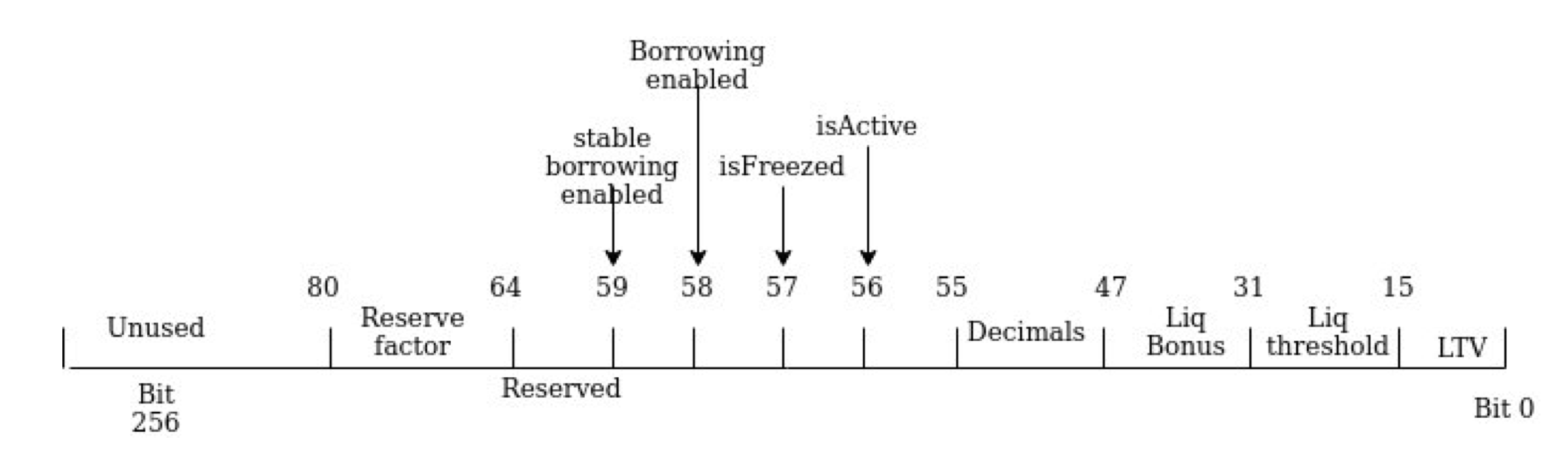
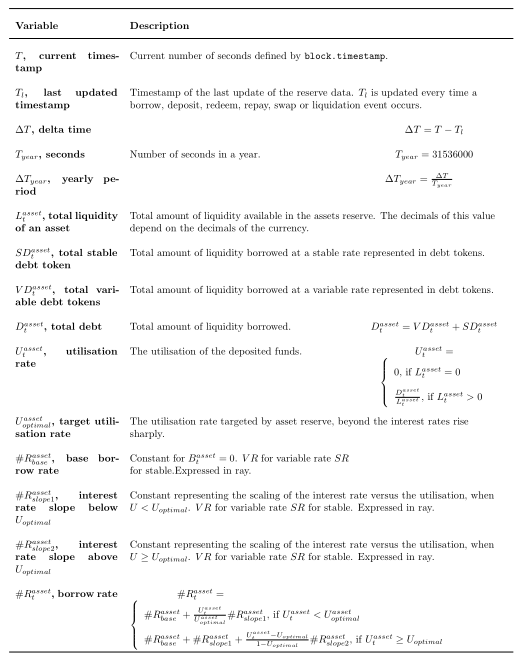


图5:储备池位掩码结构

# 5 正式的定义



表格

描述已自动生成

文本, 表格

中度可信度描述已自动生成