原文：《万物研究院：Web3的下一个十亿级用户市场？全面解读账户抽象与EIP4337的技术和应用》

作者：Steve Wang，万物研究院研究员，沃顿商学院在读，前端&合约工程师

简介

以太坊改进提案4337（EIP 4337）定义的账户抽象是一组协议层的接口。它不仅集成了web2用户熟悉的交互形式，如多因素认证（multi-factor authentication），还给web3特有的用户痛点提出了解决方案，如无需手续费（gasless）的交易。作为引入下十亿用户的协议接口，用户体验是账户抽象最关注的重点。

虽然许多现有的文章很好地解释了账户抽象，但是大多偏科普向，也有少数十分深入于技术细节。本文旨在融合两者：既提供关于账户抽象概念的全面技术解读，也分类剖析现有应用和基础设施的案例。因此本文分为三部分：在第一部分，我们将探讨EIP 4337的起源，并深入了解其技术细节，包括用户操作（UserOperation）、打包器（Bundler）、入口点合约（Entry Point Contract）、代付合约（Paymaster）、钱包工厂（Wallet Factory）和签名聚合器（Signature Aggregator）。在第二部分，我们将分析现有的市场玩家，包括智能合约钱包和第三方基础设施提供商。在第三部分，我们将讨论账户抽象的发展方向，包括该领域的创新展望和预测。

本文篇幅较长，第一部分主要为账户抽象的技术解读，只对账户抽象的应用场景感兴趣的读者可以直接跳到第二部分。

第一部分：EIP 4337的全面技术分析

首先来简要回顾一下基础知识，以太坊有两种类型的账户：外部拥有的账户（externally owned accounts，即EOA）和合约账户。

EOA是用户控制的账户，可以发送交易。EOA通过其私钥控制账户的所有权，可以通过签名执行交易，从而改变EOA的内部状态或外部合约账户的状态。由于私钥（或种子短语）是EOA所有权的唯一代表，EOA不适合定义复杂的所有权或交易逻辑

，如使用社交媒体账户登录（social login）和多方签名（multisig）的交易。EOA的局限性导致用户体验不佳：一些冗杂的步骤，如私钥和种子短语管理，更是直接阻碍了Web2用户进入Web3。大多数最受欢迎的钱包，如MetaMask，都是基于EOA的账户模型。

合约账户可以托管任意的Solidity代码，因此可以发挥EVM全部的图灵完备特性。遗憾的是，合约账户不能发送交易，因此它们的功能必须由EOA触发。智能合约钱包是一种合约账户，通过钱包提供商的EOA网络间接地被其用户触发，无论是通过中继器（relayer）还是打包器（Bundler）。

Externally Owned Account (EOA)	Contract Account (CA)
Creating an account costs nothing	Creating a contract has a cost because you're using network storage
Can initiate transactions	Can only send transactions in response to receiving a transaction
Transactions between externally-owned accounts can only be ETH/token transfers	Transactions from an external account to a contract account can trigger code which can execute many different actions, such as transferring tokens or even creating a new contract
Made up of a cryptographic pair of keys: public and private keys that control account activities	Contract accounts don't have private keys. Instead, they are controlled by the logic of the smart contract code

图：EOA与合约账户（来源：Bitcoin Insider）

智能合约钱包早在EIP 4337之前就已存在。EIP 4337的提出是为了标准化设计智能合约钱包及其相关基础设施的通用功能。EIP 4337遵循以下几个设计原则：

总体而言，所有的技术实现都在顶层（智能合约层），从而避免触及共识层和执行层等底层基础设施。账号模型这种基本设计如果在底层改动会成为像以太坊PoS主网合并一样的重要升级。因为开发人员的精力有限，只能采用合约层实现的轻量级技术路线。

协议接口的设计应该是模块化的，这样用户可以自定义选项，包括交易打包处理（Bundler）、gas代付（Paymaster）和签名聚合（Aggregator）。

理想情况下，上述每项服务都将成为一个具有竞争性的开放市场。用户可以根据提供商的价格和声誉选择最佳的服务。

EIP 4337为标准化智能合约钱包定义了六个合约接口：

首先，智能合约钱包本身，会通过打包器触发入口合约来间接触发，在链下验证交易，然后在链上执行。

其次，打包器（Bundler），用于批量验证和执行用户操作（UserOperation，即EIP 4337定义的智能合约钱包的交易类型）。

第三，入口合约（Entry Point Contract）是在以太坊只存在一份的全局合约，用于标准化交易执行并防止打包器受到恶意交易的DoS攻击。

第四，代付合约（Paymaster），用于代表钱包用户处理gas支付。

第五，合约工厂（Wallet Factory），用于标准化钱包创建的参数和流程。

第六，签名聚合器（Signature Aggregator），用于将多个交易的签名聚合为字节，以便更快地验证和执行交易。

下面我将详细介绍用户操作（UserOperation）和上述的六个合约接口，是对官方EIP 4337文档和David Philipson@Alchemy账户抽象系列的解读。

1. 账户操作（UserOperation）

UserOperation本质上和普通交易相同，只是基于EIP 4337的合约接口定义了额外的参数。这里回顾一下，一个EOA触发的普通以太坊交易有交易目标地址、转账以太坊数量、gas数量和gas价格等参数。

此外，考虑到EIP 4337合约接口的模块化设计，UserOperation包括以下主要参数用于定义接口的触发：

calldata：定义要在智能合约钱包上调用的函数签名（function signature）和输入参数，calldata在普通交易中也有

signature：验证交易确实来自某智能合约钱包地址，signature在普通交易中也有

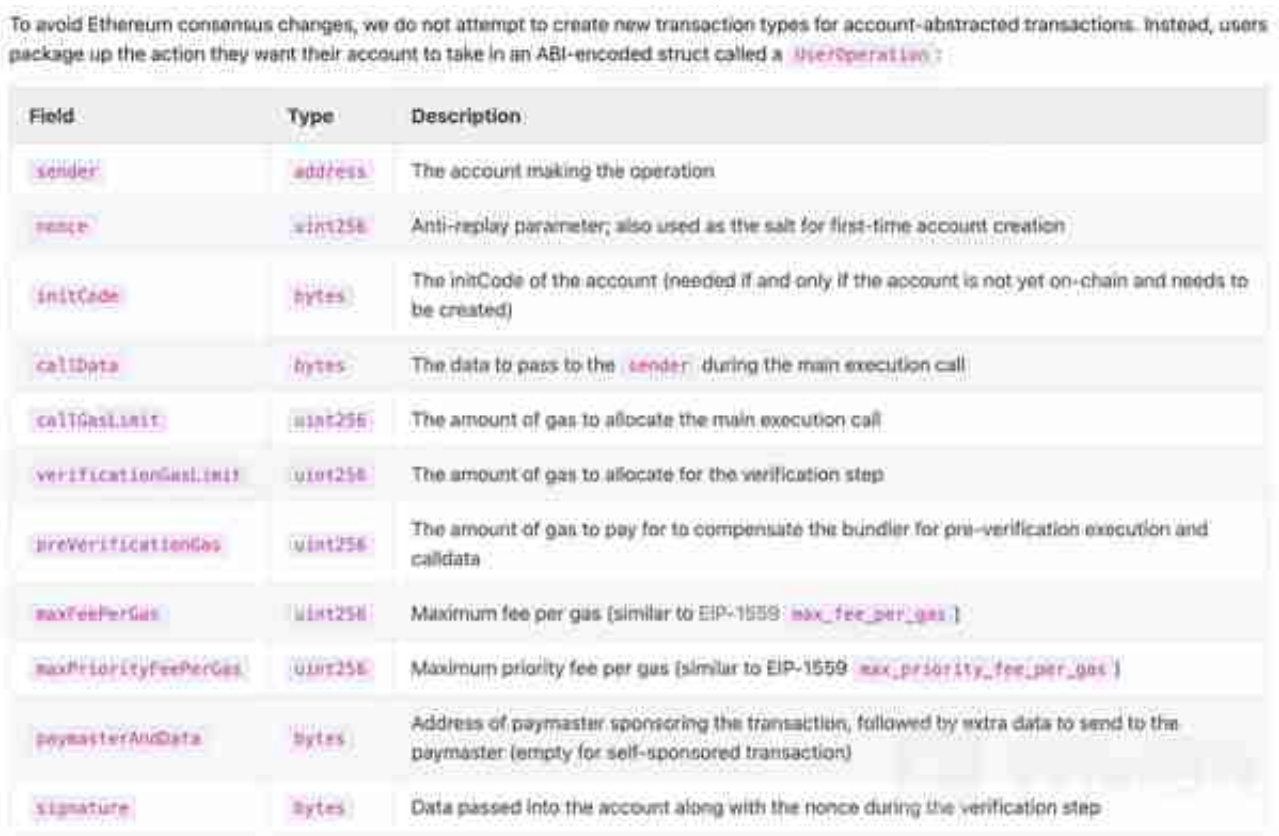
nonce：防止重放攻击，在普通交易中也有

sender：指定执行UserOperation的智能合约钱包地址

paymasterAndData：用于交易手续费抽象（gas abstraction），包含代付合约（Paymaster）地址和gas支付的具体参数

initCode：用于钱包创建，包含钱包工厂（Wallet Factory）合约地址和创建智能合约钱包的参数，如钱包的合约代码

除了需要额外的参数以外，UserOperation的工作流程与普通交易类似：它们都被广播到mempool中进行验证、执行并最终完成出块。UserOperation的验证和执行由打包器（Bundlers）触发，我将在下文中解释。



To avoid Ethereum consensus changes, we do not attempt to create new transaction types for account-abstracted transactions. Instead, users package up the action they want their account to take in an ABI-encoded struct called a `UserOperation`:

Field	Type	Description
`sender`	`address`	The account making the operation
`nonce`	`uint256`	Anti-replay parameter; also used as the salt for first-time account creation
`initCode`	`bytes`	The initCode of the account (needed if and only if the account is not yet on-chain and needs to be created)
`callData`	`bytes`	The data to pass to the `sender` during the main execution call
`callGasLimit`	`uint256`	The amount of gas to allocate the main execution call
`verificationGasLimit`	`uint256`	The amount of gas to allocate for the verification step
`preVerificationGas`	`uint256`	The amount of gas to pay for to compensate the bundler for pre-verification execution and calldata
`maxFeePerGas`	`uint256`	Maximum fee per gas (similar to EIP-1559 `max_fee_per_gas`)
`maxPriorityFeePerGas`	`uint256`	Maximum priority fee per gas (similar to EIP-1559 `max_priority_fee_per_gas`)
`paymasterAndData`	`bytes`	Address of paymaster sponsoring the transaction, followed by extra data to send to the paymaster (empty for self-sponsored transaction)
`signature`	`bytes`	Data passed into the account along with the nonce during the verification step

图：官方EIP 4337文档对UserOperation参数的定义2. 打包器（Bundler）

Bundler是一个外部账户（EOA），代表用户在智能合约钱包上验证和执行UserOperation交易，因为在以太坊上，所有交易都必须由EOA触发。Bundler使用户不用创建和记忆EOA的私钥就可以触发智能合约钱包交易，这也是智能合约钱包存在的初衷。

虽然Bundler具有很强的公共物品（public goods）属性，他们也可以获得经济上的利益，因为：

Bundler可以在执行UserOperation后将最大优先gas费（maximum priority fee）与实际gas花费之间的差额收入囊中

与普通交易的中继器（Relayer）类似，Bundler可以通过排序捆绑交易（bundle）中的UserOperation来获取MEV

尽管用户付出了成本，但Bundlers对于节省gas是有好处的，因为每执行一笔交易都需要21,000 gas的固定成本，而执行捆绑交易可以把一笔交易的固定成本分摊给捆绑交易里的多UserOperation，从而节省gas成本。此外，在同一笔交易中多次修改storage，每次额外操作的gas费都会边际递减。

需要注意的是，尽管以太坊上的交易是线性执行（非并行）以避免状态冲突（state conflict），多个UserOperations可以放在一个捆绑交易里执行，因为Bundlers可以设置同一智能合约钱包地址最多包含一个UserOperation，这样捆绑交易在修改状态时不会相互冲突。

除了这些差异之外，Bundlers与区块构建者（Block Builder）类似，因为它们都将经过验证的UserOperation广播到公共或私有的mempool中。接下来，我们将介绍入口点合约（Entry Point
Contract），Bundler需要与之交互来执行UserOperation。

3. 入口点合约（Entry Point Contract）

入口点合约是一个全局单例（global singleton）合约，所有Bundlers都需要调用它来执行UserOperation。它充当了Bundler与智能合约钱包之间的中间人：

一方面，Bundlers调用入口点合约的handleOp函数，该函数接受一个UserOperation作为输入参数。handleOp负责根据UserOperation的calldata验证并执行智能合约钱包的函数

handleOp首先在链上验证UserOperation，检查它是否由指定的智能合约钱包地址签名，以及钱包是否拥有足够的gas来补偿Bundler

如果验证成功，handleOp将根据UserOperation的calldata中定义的函数签名和输入参数执行智能合约钱包函数

另一方面，智能合约钱包需要向入口点合约存入代币用于支付gas费用给Bundlers。当Bundler使用EOA触发handleOp函数时，会产生gas费用。

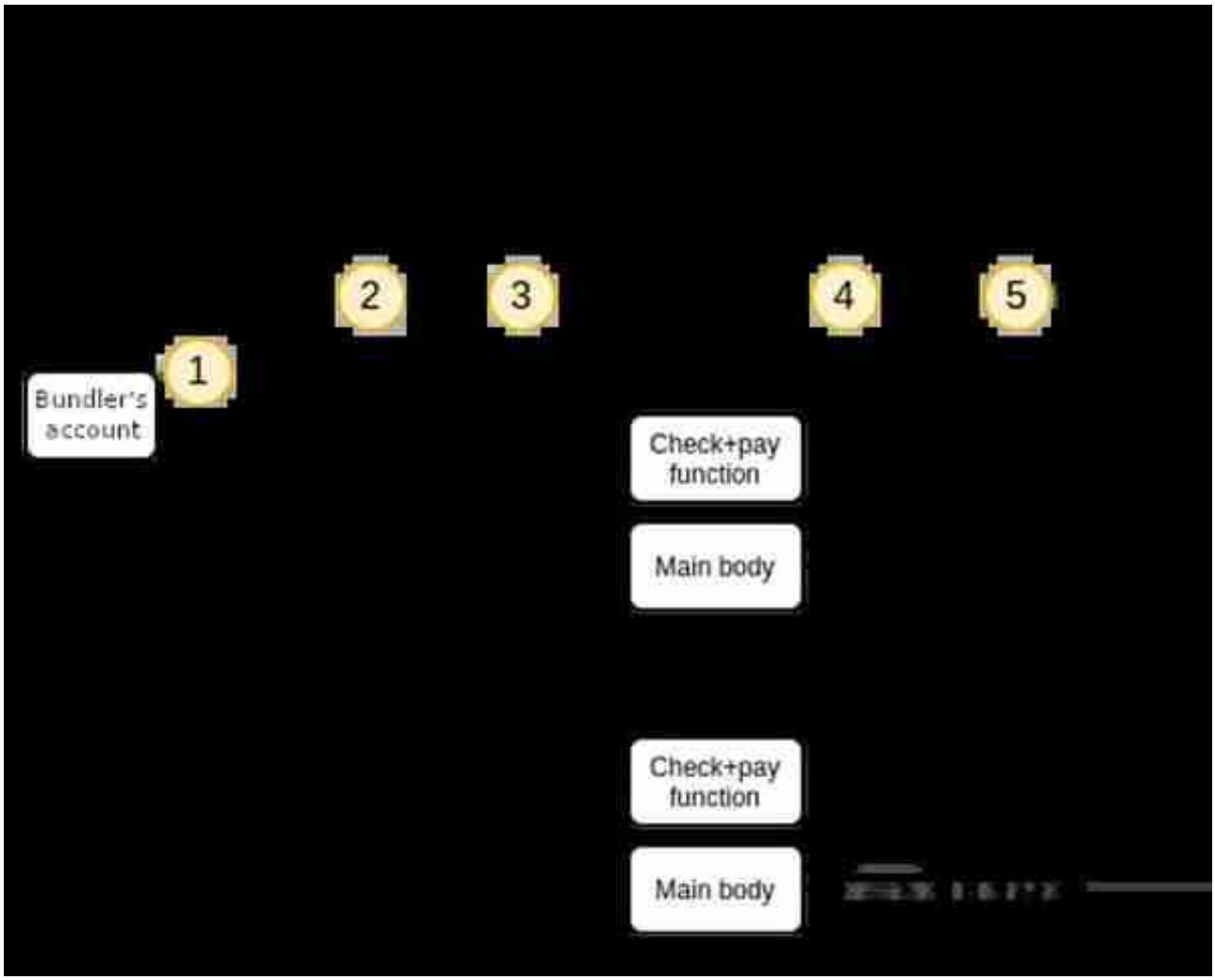
除此之外，智能合约钱包也可以用自己的账户余额支付gas费用给Bundler，也可以请求代付合约（Paymaster）代为支付，我们稍后会详细解释

没有足够gas的UserOperation在目标智能合约钱包中无法通过验证步骤，因此在执行之前就会失败

即使有足够的gas，UserOperation在执行步骤中仍可能失败，例如runtime error

无论执行是否成功，入口点合约都将支付gas费用给Bundler触发handleOp函数

入口点合约提供了智能合约钱包添加或提取代币抵押的函数



图：官方EIP 4337文档中的入口点合约工作流程4. 智能合约钱包主合约

智能合约钱包主合约将UserOperation的验证和执行步骤分开：

验证步骤在validateOp函数中定义。此函数会被调用两次：第一次是Bundler在链下模拟验证，验签UserOperation中的签名，并确保智能合约钱包有足够的gas余额；第二次是入口点合约在执行UserOperation之前进行链上验证

执行步骤在UserOperation的calldata中定义，用于指定智能合约钱包函数的签名和输入参数

通过分离UserOperation的验证和执行，Bundler可以在链下验证UserOperation，从而不需支付gas就可以过滤掉恶意交易。
因此validateOp函数也起到了链下debug API的作用。

5. 代付合约（Paymaster）

代付合约（Paymaster）定义了智能合约钱包的gas抽象的逻辑。Gas抽象的形式包括用ERC20同质化代币支付以太坊gas和无需gas的交易，两者都可以由Paymaster来实现。ERC-4337的模块化设计允许UserOperation通过paymasterAndData参数指定任意的Paymaster地址和输入参数。Paymaster具有以下功能和要求：

本质上，Paymaster是由dApp部署的智能合约，可以通过Bundler触发Paymaster的validatePaymasterOp函数以任意逻辑有选择性地为合约指定的UserOperation支付gas

Paymaster需要在入口点合约上存入以太坊用于支付UserOperation的gas

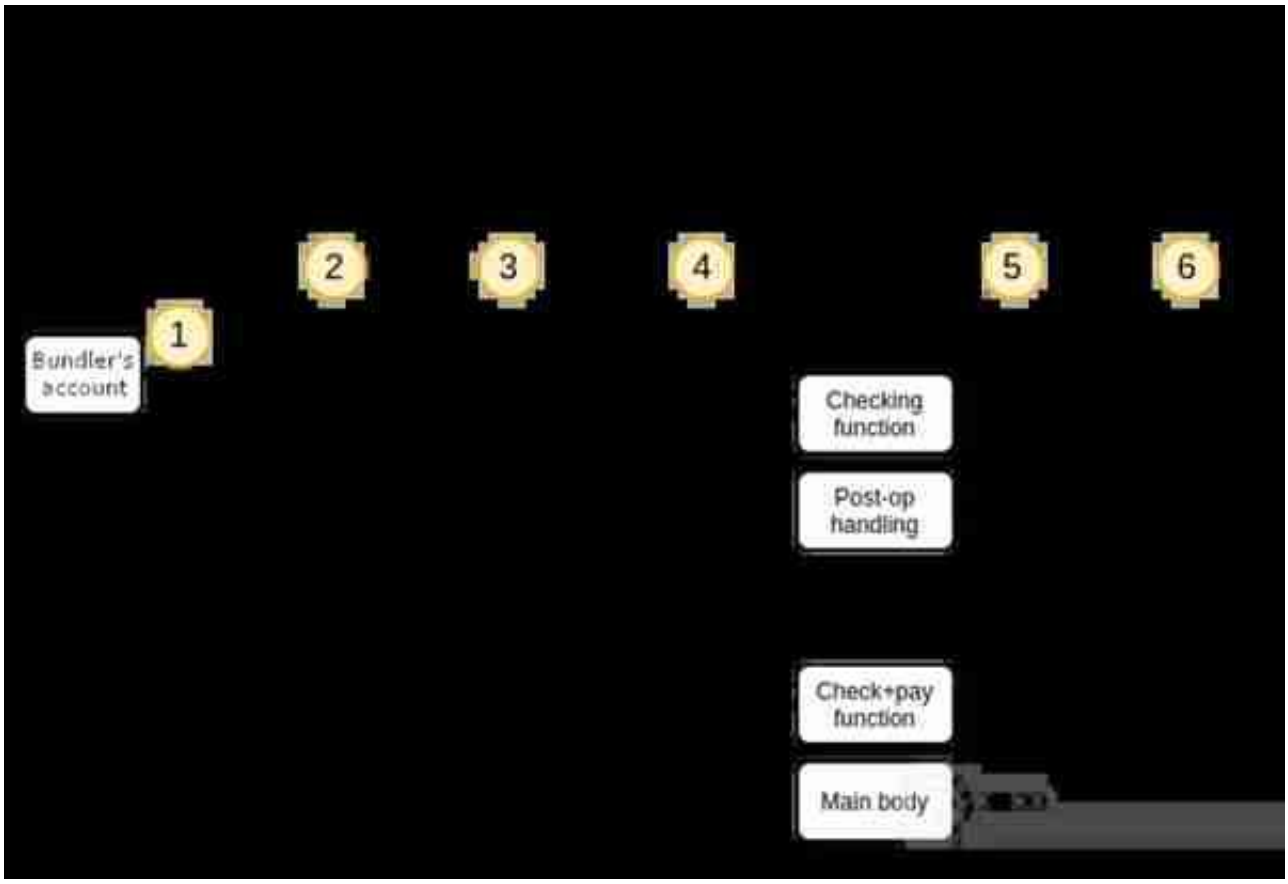
除了存入gas外，Paymaster还需要在入口点合约上额外质押以太坊。这个质押不会被slash，但可以防止机器人恶意批量创建Paymaster

Paymaster（至少在官方的愿景中）是一个充满竞争的开放市场。虽然官方没有定义Paymaster的链上信誉系统，Bundlers可以在链下记录Paymaster的服务质量并停止使用质量较低的Paymaster，从而激励Paymaster为UserOperation提供可靠的服务。

Paymaster接口定义了两个函数：

validatePaymasterOp：定义gas抽象逻辑。例如，在一个智能合约钱包以ERC20代币支付gas的场景中，此函数将确保钱包具有足够的余额。validatePaymasterOp在执行UserOperation之前被调用，符合将验证步骤和执行步骤分开的设计逻辑

postOp：在智能合约钱包执行某一函数的最后一步被调用。如果钱包没有足够的ERC20余额来支付gas费给Paymaster，postOp会取消函数执行。即使在此步骤中函数执行因为runtime error失败，gas仍然会被扣除。



图：官方EIP 4337文档中的Paymaster工作流程

图：官方EIP 4337文档中的Paymaster接口6. 钱包工厂（Wallet Factory）

Wallet Factory是一个创建智能合约钱包的合约。 回顾一下，UserOperation有一个可选的initCode字段。当initCode为空时，UserOperation只是触发智能合约钱包的功能。当initCode填充了Wallet Factory地址和新智能合约钱包的参数时，Bundler将触发相应的Wallet Factory使用指定参数创建智能合约。

Wallet Factory在以下方面有助于实现账户抽象：

用户可以通过提交填充了initCode的UserOperation来请求Bundler创建智能合约钱包，而无需拥有EOA来自行创建

用户可以通过选择具有特定定制选项的Wallet Factory并传入自己的参数来定制智能合约钱包。此外，因为Wallet Factory合约是公开的，而且受欢迎的Wallet Factory会经过全面的代码审计，所以使用Wallet Factory创建新钱包对用户更安全

用户可以选择让Paymaster代付创建钱包的gas或者用任意ERC20代币支付gas，无需拥有以太坊。此外，Paymaster可以选择只为其信赖的Wallet Factory部署的智能合约钱包支付gas

用户可以在创建钱包之前通过调用CREATE2函数获取他们的智能合约钱包地址。该函数使用触发智能合约钱包创建的EOA地址、一个salt和initCode确定性地生成智能合约地址。用户在创建钱包时不需要支付gas，只需要在用钱包进行第一笔交易时同时支付创建钱包和交易的gas，从而提高用户体验。

Wallet Factory还有其他职责。与Paymaster类似，它们需要在入口点合约上抵押ETH并持续性地为UserOperations提供良好的服务，以便从Bundler那里获得更多流量。

7. 签名聚合器（Signature Aggregator）

因为不同的智能合约钱包使用不同的签名算法，所以需要先将使用相同签名算法的UserOperations聚合起来，然后分别进行验证。此外，由于链上密码学计算会消耗大量gas，支持聚合的签名方案（如BLS）可以在链上验证过程中节省gas。因此我们需要一个名为签名聚合器（Signature Aggregator）的EIP 4337智能合约接口。它可以通过实现以下两个函数来支持特定的签名算法：

aggregateSignatures：输入参数为一个UserOperations数组，并使用特定的签名算法输出一个聚合签名

validateSignatures：输入参数为一个UserOperations数组和一个聚合签名，用于对数组里的所有UserOperation进行验签

与其一次验证一个UserOperation，Bundler首先会使用多个签名聚合器合约（每个签名算法使用一个）生成多个聚合签名（每个签名算法生成一个）。之后，Bundler将UserOperation数组、聚合签名和聚合器地址传递给入口点合约，每个UserOperation数组会调用其对应的签名聚合器的validateSignature函数。验证通过后，Bundler就会在智能合约钱包上执行这组UserOperations。

与Paymaster和Wallet Factory类似，聚合器也需要在入口点合约上质押以太坊并

保持为UserOperations提供良好服务的记录。

```
interface IAggregator {

  function validateUserOpSignature(UserOperation calldata userOp)
  external view returns (bytes memory sigForUserOp);

  function aggregateSignatures(UserOperation[] calldata userOps) external view returns (bytes memory aggregatesSignature);

  function validateSignatures(UserOperation[] calldata userOps, bytes calldata signature) view external;
}
```

图：官方EIP 4337文档中的签名聚合器接口8. 整体工作流程总结

EIP 4337的模块化设计将智能合约钱包的账户抽象功能分为多个接口。这些接口的功能合在一起可以总结为如下的工作流程：

用户向Bundler发送UserOperation

如果UserOperation填写了initCode参数，Bundler会触发Wallet Factory来创建有确定地址的新钱包

如果UserOperation填写了paymasterAndData参数，Bundler会收取Paymaster在入口点合约上质押的以太坊来支付gas。如果没有填写paymasterAndData，Bundler会收取智能合约钱包在入口点合约上质押的以太坊来支付gas

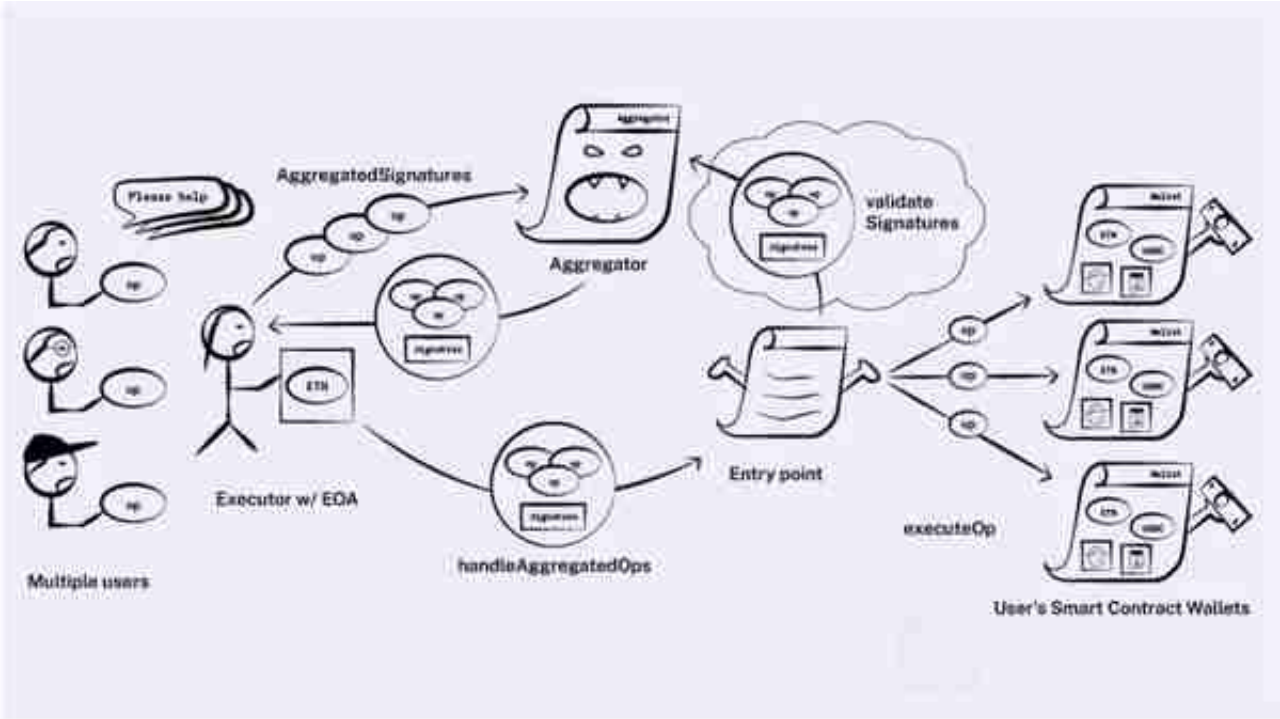
Bundler可以选择性地使用签名聚合器

Bundler会使用validateOp、validatePaymasterOp和validateSignatures函数在链下模拟验证UserOperation，确保签名正确且UserOperation有足够的gas

如果链下模拟验证通过，Bundler将使用上述函数在链上验证UserOperation

如果链上验证通过，Bundler将在链上执行UserOperation，无论执行是否成功，都会扣除gas

图：没有签名聚合器的账户抽象工作流程（Alchemy提供）



图：带签名聚合器的账户抽象工作流程（Alchemy提供）9. 其他注意事项

UserOperation在链下和链上都会触发validateOp等函数进行交易验证。为了防止智能钱包合约外部的状态变化（state
change）导致链下和链上的验证结果不一致，EIP 4337禁止验证函数使用读取合约外部storage和读取全局状态（如时间戳）的OpCodes。这种禁止适用于validateOp、validatePaymasterOp和validateSignatures这些验证函数。

第二部分：账户抽象市场参与者1. 智能合约钱包及配套SDK

智能合约钱包是账户抽象市场中最大的参与者。为了与EIP 4337兼容，它们通常会实现自己的打包器（Bundler）、代付合约（Paymaster）、钱包工厂（Wallet Factory）和签名聚合器（Signature Aggregator）。这些钱包还配备了方便dApp集成钱包的SDK。智能合约钱包市场比较拥挤，既有老牌大玩家如多签钱包Gnosis Safe在原有产品基础上实现账户抽象功能，也有市场新人如Candide专注于构建完全兼容EIP
4337的智能合约钱包。本节将介绍这些钱包的共同特点以及每个特点中的差异化。

a. 社交登录（social login）和社交恢复（social recovery）

社交登录为智能合约钱包提供了Web2用户非常熟悉的交互体验，例如使用已有的

Google帐户创建和登录钱包。同样的逻辑也可以延伸到社交恢复上，例如使用邮箱重置智能合约钱包的密码。社交登录和社交恢复的逻辑一般在钱包的主合约中定义。

这里我们引入守护者（Guardian）的概念。守护者是指可以授权用户访问某账户或帮助用户重制恢复某帐户的实体。守护者可以是Web2账号也可以是邮箱地址。这个概念在Web2中已经有了很广泛的应用，现在也可以通过账户抽象的概念在智能合约钱包中实现。

一个智能合约钱包可以有多个守护者。用户需要在创建钱包过程中或之后设置守护者，并达到一定的守护者验证阈值，例如3个守护者中的2个，以登录或恢复钱包。这个过程通常被称为多因素认证（multi factor authentication）。以下是常见的智能合约钱包守护者类型：

Web2服务提供商：例如钱包用户的社交媒体帐户，通常通过实施OAuth（Open Authorization，即开放授权）标准来实现。该标准允许钱包获取用户授权来访问由另一个Web2应用托管的用户账号信息

用户设备：例如浏览器存储和移动端存储

电子邮件：例如通过点击可以发送签名的电子邮件链接进行授权

多签名：用户可以设置多个个人拥有的EOA或智能合约钱包作为守护者

MPC：用户可以将私钥分割成多个份额，每个份额由MPC网络中的一个节点控制，且不会泄露完整的密钥

SWIE（sign in with Ethereum，即用以太坊登录）

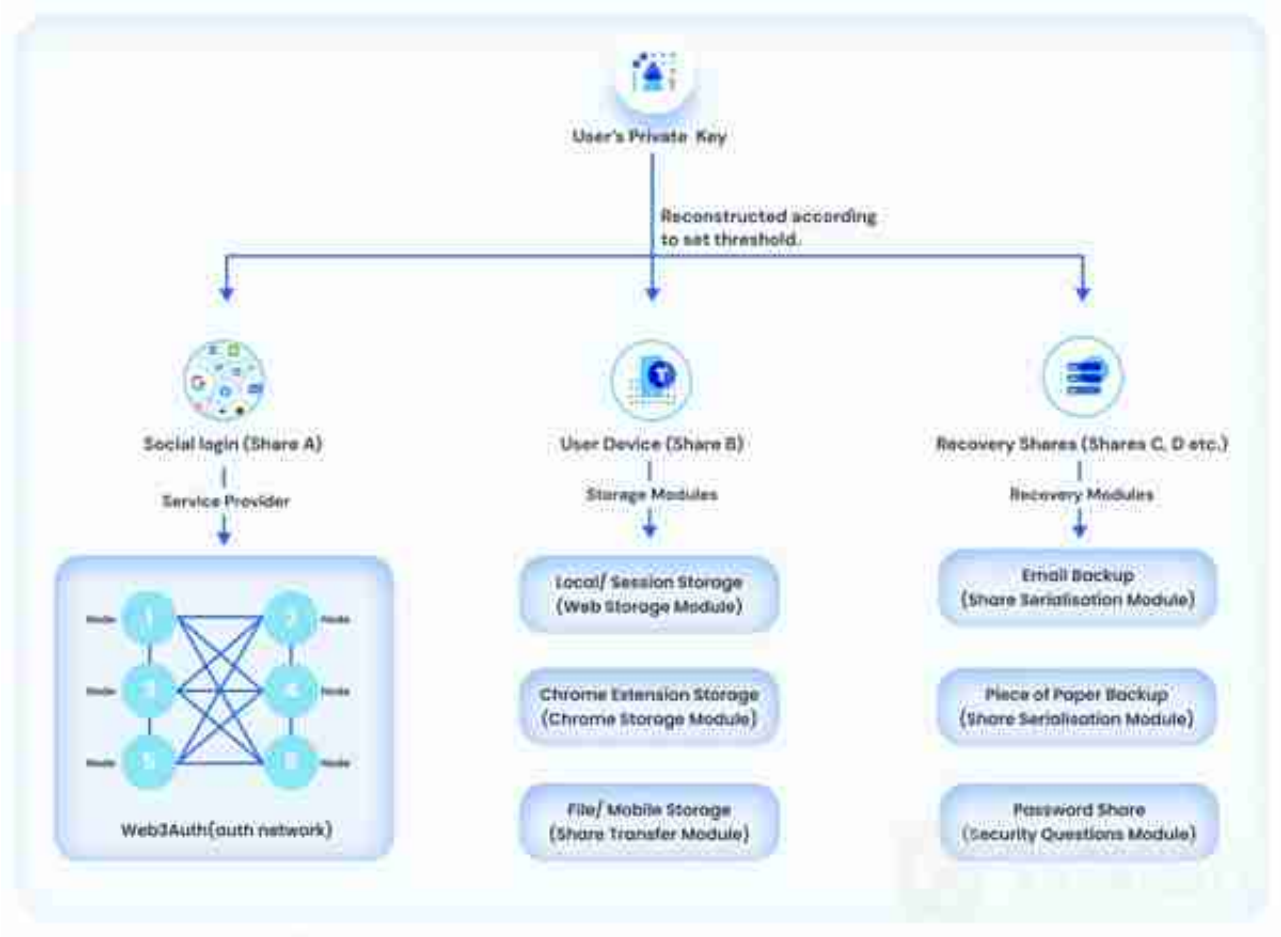
让我们看看一些市场上玩家对此功能的实现：

Web3Auth是许多高流量智能合约钱包使用的社交登录和MPC服务，这些钱包包括Biconomy、Etherspot和0xPass。Web3Auth把用户密钥分割成多份给社交登录、用户设备和其MPC节点网络等守护者。产品逻辑实现完全在链下，且没有任何一个守护者会存储完整的私钥。（请参见下图。）

BLS Wallet和Argent使用多签技术（multisig）通过用户指定的多个EOA地址进行密钥恢复。

UniPass实现了一种创新的方法，通过电子邮件进行社交恢复。用户在链上提交守护者的邮箱，并通过智能合约在链上验证邮箱的DKIM签名。DKIM（Domain Keys Identified Mail，即域名密钥识别邮件）是一种电子邮件身份验证协议，用于创建数字签名。邮箱供应商可以使用该数字签名验证邮件发送者的身份。UniPass还使用零知识证明对链上的用户信息进行脱敏处理。UniPass也支持邮件之外的社交恢复方法。

Soul Wallet在链上验证守护者的邮箱地址以进行社交恢复。



图：Web3Auth社交登录和社交恢复，此案例中有三个守护者

b. Gas Abstraction（gas抽象，gas即交易手续费）

Gas abstraction包含无gas交易和使用任意ERC20代币支付gas。这些逻辑可以在代付合约（Paymaster）中实现，也可以通过中继器（Relayer）实现。

许多智能合约钱包自己实现了与EIP
4337兼容的Paymaster合约，并在入口点合约上质押代币，帮助用户支付gas。

Relayer是在EIP 4337之前就存在的另一个gas abstraction的解决方案。 Relayer是用户信任的交易转发者，执行一种称为“元交易”（meta-transaction）的特殊类型的无gas交易。元交易是一种以太坊交易，它在原始交易中插入另一个交易。用户使用私钥对元交易进行签名，并将其发送给Relayer，后者对其进行验证，将其提交给区块链，并代付gas。Relayer只是执行交易，而不改变交易本身。 Relayer会获取经济上的利益，因为它们会向执行交易的合约收取gas费用加上一小笔利润。例如，Relayer可以向支持无gas交易的去中心化应用（dApp）收费。Relayer可以是中心化的可信第三方，也可以是去中心化的网络。Relayer与代付合约（Paymasters）的区别在于以下几点：

除了支付gas外，Relayer还会对交易进行签名，类似于EIP 4337中的Bundler所做的事情

Relayer不只处理UserOperations类型的交易

Relayer不在入口点合约上质押gas

让我们看看一些市场上玩家对此功能的实现：

Biconomy不仅自行实现了与EIP 4337兼容的Paymaster，还有一个支持任意ERC20代币支付gas的Relayer网络。

Argent使用第三方Relayer以元交易的形式支付gas。

Candide自行实现了Paymaster。

UniPass使用自己的Relayer节点支付gas，计划在未来添加“观看广告免gas交易”模式并支持使用跨链桥（bridge）支付gas。

到目前为止，大多数ERC20支付gas的解决方案（如Candide和Soul Wallet）仅支持非常有限的代币类型，主要是稳定币，尽管我预见这在未来会发生变化。

图：Biconomy 使用Paymaster支持无gas交易（基于EIP 4337的技术实现）

图：Biconomy 通过Relayer使用任意ERC20支付gas（并非基于EIP 4337的技术实现）

c. 法币出入金通道（ramps）和跨链桥（bridges）

许多现有的钱包，如MetaMask，已经通过和第三方供应商合作的方式把法币出入金通道和跨链桥集合在钱包中。这些出入金通道和跨链桥可以进一步与gas abstraction中的代付合约（Paymaster）进行集合。

让我们看看一些市场上玩家对此功能的实现：

Biconomy和0xPass与Transak合作提供法币通道。Biconomy还提供跨链桥和跨链通信服务。

Argent Vault与Moonpay、Transak和Wyre合作提供法币通道，并具有内置的DeFi协议聚合器。

Etherspot、UniPass和Braavos支持法币通道、swap和跨链桥。

d. 交易批处理（transaction batching）

交易批处理是智能合约钱包独有的功能，允许钱包用户在一个链上交易中执行多个交易。一种实现方法是调用一个multiCall函数，它接受两个参数：一个calldata数组（每个calldata定义一个函数调用）和一个合约地址数组（每个函数调用的合约地址）。multiCall函数中有一个循环（loop）并在每次循环里执行一个函数调用。交易批处理可以使多个交易分摊一个以太坊交易的固定gas费，从而减少成本。

请注意，交易批处理不同于签名聚合（signature aggregation），后者接受一个UserOperations数组作为输入参数并输出一个聚合签名字节以加快多个签名验证的速度。而这个数组中的每个UserOperation都可以调用一个multiCall来执行交易批处理。

在EIP 4337中实现交易批处理的方式如下：入口点合约调用handleOp函数，然后该函数再调用智能合约钱包主合约内定义的multiCall函数。

让我们看看一些市场上玩家对此功能的实现：

Biconomy设置了一个MultiCall合约，智能合约钱包可以调用这个合约来批量执行交易。

Argent通过名为Transaction Manager的智能合约模块来批量执行交易，该模块支持multiCall和用户会话密钥（session key）。

Etherspot、Candide、Openfort和0xpass都支持交易批处理。

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract MultiCall {
    function multiCall(
        address[] calldata targets,
        bytes[] calldata data
    ) external view returns (bytes[] memory) {
        require(targets.length == data.length, "target length != data length");

        bytes[] memory results = new bytes[](data.length);

        for (uint i; i < targets.length; i++) {
            (bool success, bytes memory result) = targets[i].staticcall(data[i]);
            require(success, "call failed");
            results[i] = result;
        }

        return results;
    }
}
```

图：示例 MultiCall 合约（Solidity by Example提供）

e. 智能合约钱包的模块化设计（modular design）

智能合约钱包相对于EOA的一个主要优势是其模块化设计。钱包的功能可以通过钱包提供商开发的合约模块进行扩展。这些模块提供了EIP 4337接口中未定义的功能，并可以由用户自定义。模块化设计下，智能合约钱包是可升级的。这种做法在EIP 4337发布之前就已经是行业标准，由诸如Gnosis Safe之类的产品率先推出。Gnosis Safe是一款主要针对商业用户的多签名钱包，提供一系列EIP 4337没有定义的功能：

支出限制：限制有签名权限的帐户可以从钱包中提取的金额。

定期打款：根据自定义时间自动打款。

角色和授权：为钱包不同的交易和行为定义所需的角色和授权。

组织权限：基于组织内不同角色的权限设置。

白名单和黑名单

同样的模块化设计也被EIP 4337兼容的钱包继承，包括Argent、Candide、Soul Wallet和zeroDev。

另一种扩展智能合约钱包功能的方法是在钱包里原生地集成去中心化应用（dApps）。例如，Gnosis Safe为dApps提供了一个SDK。使用此SDK的dApp可以直接出现在钱包界面中，等同于为钱包提供了一个内置的应用商店。其他类似解决方案包括WalletConnect。然而，这些前端解决方案并不是协议层的设计，因此超出了本文讨论的范围。

图：Candide钱包的模块化设计与Safe（即Gnosis Safe）类似（Candide钱包官网提供）2. 基础设施提供商

下面将讨论各种Layer 2对账户抽象的支持以及打包器（Bundlers）和代付合约（Paymasters）的第三方基础设施提供商。

a. Layer 2上的账户抽象

鉴于最近L2的爆炸式增长，许多智能合约钱包已将开发重点转向L2。许多L2都原生地支持与EIP 4337非常相似的账户抽象合约标准，本节将对此进行探讨。

Optimism

Optimism的第一版通过引入三个EVM不支持的OVM OpCodes来实现账户抽象。在Optimism的技术实现中，智能合约是唯一的账户类型（不存在EOAs），因此所有用户钱包都是智能合约钱包。智能合约钱包还可以通过调用upgrade函数升级合约代码。

不幸的是，为了与EVM保持完全一致和出于安全方面的考虑，Optimism的第二版移除了这三个OVM Opcodes和对账户抽象的支持。

虽然目前有少数智能合约钱包正在Optimism上构建，但是尚无关于支持账户抽象的官方声明。

Arbitrum

虽然目前有少数智能合约钱包正在Arbitrum上构建，但是尚无关于支持账户抽象的官方声明。

Starknet

与以太坊不同，Starknet不区分EOA和合约账户。因此，所有Starknet账户都是智能合约账户。Starknet的账户模型受到EIP
4337的很大影响，具体表现在以下方面：

所有Starknet智能合约账户都必须包含validate和execution函数。validate是必需的函数，通过验证签名确保只有账户所有者才能发起交易，同时保证交易执行者可以获得足够的gas费。用户可以在validate和execute函数中实现任意逻辑，从而灵活扩展账户的各种功能。例如，用户可以在validate函数中实现不同的验签算法。

为了防止交易通过验证但无法执行，Starknet禁止validate函数调用外部合约的状态（state）。

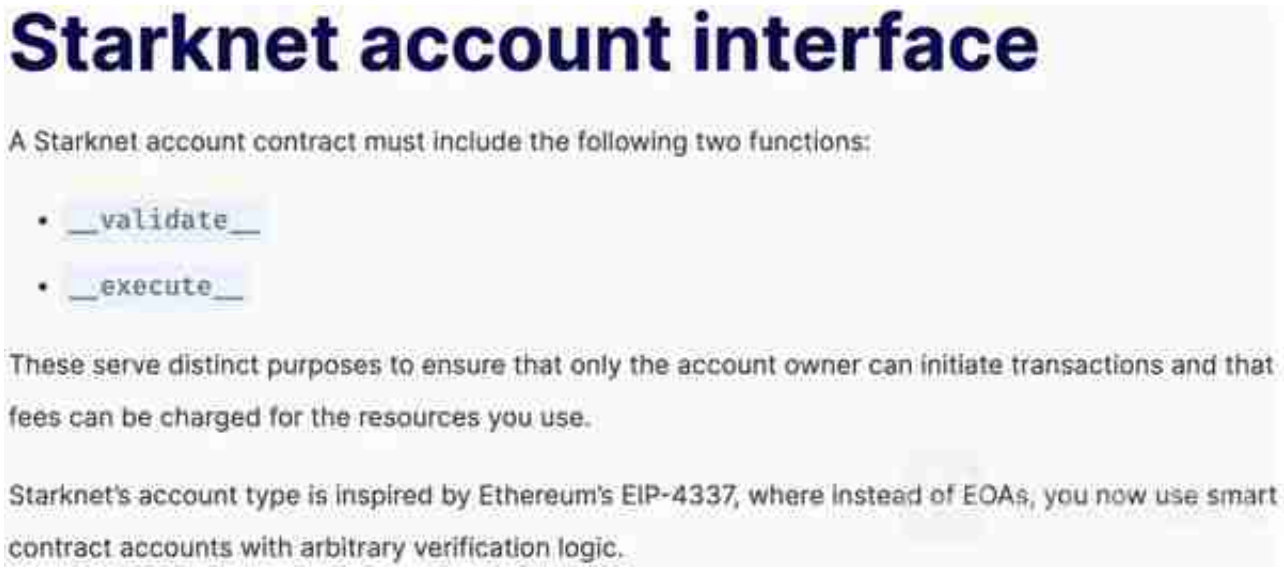
Starknet对账户抽象的实现和以太坊相比依旧存在显著的差异：

Starknet不区分常规交易和UserOperations，因为所有交易都是由合约账户触发的。在以太坊中，Bundlers执行UserOperation交易，而在Starknet中，序列器（Sequencer）执行所有交易。

Starknet尚未实现类似Paymaster的交易手续费抽象协议。

Starknet要求有代币余额的账户通过调用一个专门的deploy_account函数来创建新的合约账户。而在EIP 4337中，Bundler通过执行initCode参数不为空的UserOperation交易来部署合约账户，部署过程不必须有代币余额的账户，因为gas费可以由Paymaster代付。

如果通过验证的交易在执行步骤失败，Starknet的序列器将无法抽取任何gas费，而以太坊没有这个问题。



图：Starknet没有EOA账户类型，只有合约账户（Starknet官网提供）

zkSync

zkSync也不区分EOA和合约账户，因此原生地支持账户抽象。zkSync的账户模型与EIP 4337十分类似，具体表现在以下方面：

账户（智能合约钱包）接口也将validateTransaction和executeTransaction函数分开。

代付合约（Paymaster）接口也包括validateAndPayForPaymasterTransaction和postOp函数。前者定义了Paymaster代付交易的逻辑。后者可以确保在交易执行后，Paymaster能够抽取gas费补偿，尽管这部分功能尚未实现。

用户也通过填写交易中的paymaster地址和paymasterInput两个参数来调用Paymaster，类似于EIP 4337中的UserOperation。

然而，zkSync实现的账户抽象和以太坊也存在显著差异：

zkSync允许validateTransaction函数调用已部署的外部合约，因为已部署的合约在zkSync中是不可更改的（immutable）。
以太坊禁止验证函数调用外部合约，以防止状态更改（state change）造成交易验证通过而交易执行失败。

zkSync允许validateTransaction调用发出此交易合约账户的外部存储，例如合约账户在外部合约上的代币余额，理由同上。

在交易验证期间，Paymaster可以调用外部存储，理由同上。

b. 第三方基础设施提供商（非智能合约钱包）

如上所述，账户抽象基础设施包括可以发送UserOperations的客户端SDK、打包器（Bundlers）、代付合约（Paymasters）和签名聚合器（Signature Aggregator）。这些基础设施通常由智能合约钱包自身实现，从而更好地整合产业上下游，但是大多数都不是为第三方使用而设计的。因此市场上需要第三方基础设施提供商来提供模块化和可定制的Bundler和Paymaster服务。一些知名的基础设施玩家比如Alchemy也进入了这个市场。本节会深入探讨这些第三方基础设施提供商。

Stackup

Stackup是一个Bundler和Paymaster的提供商。其Bundler采用Go语言实现，通过了所有EIP 4377测试套件的要求，完全开源且免费使用。Stackup Bundler支持不同的模式：

私有模式（private mode）：UserOperations进入一个私有内存池（mempool），以更慢的交易执行速度换取更好的隐私。UserOperation不会显示在公共内存池中，从而避免了MEV攻击。

搜索模式（searcher mode）：由以太坊生态的机器人运营者（即bot operator，也称为searcher，例如DeFi套利机器人）使用，并与诸如Flashbot之类的区块构建者（block builder）集成，通过MEV-Boost发送UserOperations，允许searcher为特定的交易排序出价，以供block builder选取MEV最大的交易排序构建区块。

Stackup还支持两种类型的Paymaster：

Verification Paymaster：提供需要链下交易（如法币出入金）的gas抽象。例如，用户可以选择使用信用卡订阅Paymaster服务来支付gas费。用户也可以定制gas代付的逻辑。Stackup将通过“按需付费”（pay as you go）模式向用户收费。

Deposit Paymaster：允许用户用ERC20代币支付gas费。

图：Stackup的基础设施产品套件

Blocknative

Blocknative主要是一个内存池浏览器和可视化工具提供商。得益于其对于内存池较强的理解，Blocknative提供了具有独特功能的打包器（Bundler）服务，例如：

使用EIP 4337交易浏览器（EIP 4337 UserOps Explorer）可视化内存池中UserOperations的状态，方便钱包和dApp实时监控交易的状态

Blocknative的核心产品内存池浏览器也可以监听入口点合约中待处理和已确认的Bundler交易

Alchemy

Alchemy目前正在开发其Bundler和Paymaster服务，潜在用户可以加入waitlist。

图：Alchemy的EIP 4337基础设施服务还在开发中

eth-infinitism

eth-infinitism是以太坊基金会的官方团队，目前已经实现了EIP 4337标准定义的Bundler和Paymaster。关于该团队的文档很少，但是根据Stackup的一篇文章，截至2023年2月，eth-infinitism是通过EIP
4337官方测试套件的唯一两个Bundler之一（另一个是Stackup）。

第三部分：账户抽象的未来之路1.
归根结底，账户抽象市场的生死存亡取决于生态对EIP 4337的采用，而这个市场还非常早期

根据Web3Caff Research最近的一篇研报，以太坊中所有交易的发起地址（from参数）去重后的总数约为1.5亿。然而，智能合约钱包的总数，以Gnosis Safe和Argent账户的总和来估算（这两个产品的用户最多），仅为15万。

假设在终局，EOA和智能合约钱包的数量相等，我们可以预测智能合约钱包的市场规模有1000倍的潜力。 尽管如此，生态对智能合约钱包的采用并不会一帆风顺。像MetaMask这样的EOA钱包巨头仍未宣布采用EIP 4337的计划。尽管我们不知道他们的意图，但也不难推断，主流EOA钱包的巨大利润并不足以支持其团队内部采用一个基础设施都不完善的新标准。UserOperation仍然不是dApps广泛使用的主流交易类型。虽然如此，我依旧对智能合约钱包市场保持乐观态度，也可以预见到智能合约钱包对用户体验的大幅改善会为账户抽象市场带来的指数级的增长，尽管这种爆发可能还需要一两年才发生。

2. EIP 4337尚未完成，同志仍需努力

尽管EIP 4337定义了诸如打包器（Bundler）、代付合约（Paymaster）和签名聚合器（Signature Aggregator）等基础设施的合约接口，以太坊官方并没有给出很多重要技术问题的解决方案，而这些问题需要项目方反复尝试不同的技术实现，比如：

目前Bundler主要在私有内存池（private mempool）广播交易，UserOperation被直接发送给指定的区块构建者（block builder）。Bundler网络是否有可能使用公共内存池（public mempool）？

Bundler能否通过对UserOperations进行排序来抽取MEV？区块构建者能否通过对捆绑交易（bundle）进行排序来抽取MEV？MEV将如何在Bundler和区块构建者之间分配？我们是否应该将Bundler和区块构建者分开？

考虑到Bundler需要完成大量的链下模拟和链上验证，它们能否同时作为可靠的区块构建者？

UserOperation的内存池和常规交易的内存池应该如何协调以避免状态冲突？

3. L2采用EIP 4337的速度各异

尽管zkSync和Starknet已经原生地支持账户抽象，但它们仍未实现所有的EIP 4337接口，并且在技术实现上与以太坊存在差异。尽管许多项目团队正在为Optimism和Arbitrum构建智能合约钱包、打包器（Bundlers）和代付合约（Paymasters），但这些L2尚未宣布官方支持EIP 4337的计划，账户抽象也并非他们的开发重点。

从纯技术的角度来看，L2账户抽象基础设施的实现比L1更为复杂。例如，在验证步骤中，一个L2智能合约钱包需要同时估算L1和L2的gas费，并分摊给捆绑交易（bundle）里的每个UserOperation。

4. 抛开技术细节，一个优秀的智能合约钱包需要有哪些特点？

尽管钱包市场上有许多创新的功能（如gas抽象、社交恢复、MPC等），大多数智能合约钱包都支持所有上述功能，因为这些功能的技术实现并不困难。因此，智能合约钱包的商业层面与用户体验层面同样重要。重要的商业因素包括：

与dApps合作：与每个L1或L2链的流量入口级dApps合作，尤其是基础设施类的dApps（如稳定币或DeFi协议），是吸引用户的主要途径

实用功能：如钱包内部集成的NFT市场、launchpad或快速集合新的交易对

外部支持：如来自VC或以太坊基金会的官方支持，这些支持可以帮助钱包找到第一批用户

5. 一个优秀的打包器（Bundler）需要有哪些特点？

无需许可（permissionless）且模块化（modular）的Bundler服务是一种公共物品（public good）。绝大多数Bundler的开源性质决定它们是非排他性和非竞争性的。任何RPC端点都可以通过复制开源代码来运行一个Bundler。即使运行Bundler的RPC端点通过API密钥来收取服务使用费，Bundler服务也比其他基础设施（如代付合约Paymaster）更难盈利，因为后者可以通过与第三方出入金或DeFi协议聚合器提供商合作轻松赚取费用差价。虽然更难商业化，但是Bundler是一种重要的公共物品，因为UserOperation的验证和执行需要尽可能多的Bundler参与从而更好地去中心化。因为Stackup和eth-infinitism是目前唯二可用的第三方Bundler服务提供商，所以我们肯定需要更多这样的Bundler服务提供商。

Bundler服务提供商需要克服许多技术难点：

对链下和链上的多个验证步骤进行深入研究，从而确保UserOperation可以成功执行。技术难点包括理解多个UserOperations之间可能出现的状态冲突（state

conflict）。

理解验证函数对某些OpCode的禁用：

由于读取可变的外部状态（external state）可能导致交易验证成功但执行失败，因此EIP 4337禁止验证函数调用读取外部状态的OpCodes。

不同的L2对读取外部状态的OpCode有略微不同的禁用，会对多链开发造成挑战。

正在构建Bundler的项目方需要通过traceCall函数来获取每个函数调用的OpCodes。然而，大多数第三方RPC节点都不支持traceCall，因此Bundler项目方可能需要运行自己的节点，这也存在技术壁垒。

目前大多数Bundler项目方的开发重点都在L2，需要分别计算L1和L2的gas费。

实现私有和公共内存池的P2P network。私有内存池需要为searcher和dApps提供定制化的服务，例如白名单。公共UserOperation内存池的行业标准尚未完成。

Bundler服务提供商的差异化在于：

Bundler服务是专为钱包构建的，还是第三方基础设施提供商构建的？

Bundler只是钱包项目方需要构建的众多事物之一，因此他们更可能专注于为钱包构建一个最基础的可用的Bundler。

第三方基础设施提供商需要构建无需许可和模块化的Bundler，并为不同的场景提供优质的服务。

类似于以太坊节点，Bundler服务采用不同的语言实现。这种语言多样性可以防止单点故障，对以太坊生态有益。

对私有和公共内存池的支持以及对私有内存池进行定制的选项。

对L1和不同L2链的支持，每个链在接口实现上略有不同。

6. 让我们聊聊创新

本节将列举智能合约钱包和账户抽象基础设施中最需要的创新。

无需许可和模块化的EIP 4337基础设施

让我们首先设想一下EIP 4337基础设施的终局：

许多打包器（Bundler）、代付合约（Paymaster）和签名聚合器（Signature Aggregator）提供商会在一个开放市场上竞争。用户可以用最低费用获取高质量服务

市场上既存在无需许可的Bundler、Paymaster和Signature Aggregator公共端点，也存在专门为某个智能合约钱包或dApp部署的端点

智能合约钱包、dApp和个人用户可以以无需许可（permissionless）、模块化（modular）和用户友好的方式部署上述的基础设施。换句话说，第三方基础设施提供商允许从任意地址快速且可定制地部署Bundler、Paymaster和Signature Aggregator

在当前市场状态下，许多智能合约钱包已经构建了自己的基础设施，但是这些基础设施并未针对第三方的不同使用场景。诸如Stackup之类的第三方基础设施提供商正在开发模块化的Bundler和Paymaster，但是它们的部署过程仍然远未达到无需许可。因为EIP 4337尚未完成，这些基础设施的模块化功能还没有被全部定义。此外，由于UserOperation交易量依旧较低，启用了签名聚合的钱包（如BLS Wallet）仍然不是主流。

EIP 4337下游基础设施

这些包括EIP 4337未定义但实现EIP 4337基础设施所需的下游基建，比如可以集成到Paymaster的法币出入金聚合器和DeFi聚合器。因为这些解决方案已经存在，所以创新最有可能在现有提供商内部发生，提供商只需重新配置它们公开的接口。

dApp SDK和脚手架

尽管现有的智能合约钱包都为了方便dApp集成而构建了自己的客户端SDK，但市场仍然缺少：

支持账户抽象功能的类似于ether.js的dApp开发标准库，比如对如下功能的封装：

使用Web2社交媒体账户和电子邮件创建钱包和恢复钱包地址

UserOperations的创建、签名、发送以及的事件监听

快速部署Paymaster和Signature Aggregator

聚合所有主要智能合约钱包的SDK和UI工具

用于快速前端部署的脚手架工具，以便dApp开发人员可以专注于他们产品的业务逻辑

由账户抽象赋能的新型dApp的模板，例如支持gas抽象的游戏

更激进的做法：将账户从智能合约钱包分离

EIP 4337的智能合约钱包接口只需要一个validateOp函数，将所有的核心业务逻辑，如用户注册、权限控制和社交恢复等，留给钱包开发者自由决定。虽然这提供了更多的灵活性，但不同的智能合约钱包会更加割裂以太坊生态，因为一个钱包中的数据不能轻易地转移到另一个钱包。dApp也可能通过支持某些智能合约钱包而不支持其他钱包加剧这种分裂。

此外，尽管用户可以通过邮箱或其他在Web2常见的流程登录智能合约钱包，他们仍需要通过“连接钱包”这类Web2用户不熟悉的操作流程来登录dApp。

解决这些问题的一个激进方案是通过创建新的账户标准，将账户与智能合约钱包分离，例如Hexlink倡导的EIP 6662。EIP 6662定义了一个IAccountMetadata的账号接口，使用户可以自定义自己账号的认证方法（如Web2社交媒体账号），从而将认证逻辑从钱包主合约转移到单独的账户接口。支持此接口的智能合约钱包可以允许用户将其账户数据转移到另一个钱包。支持此接口的dApp可以允许用户使用账户接口中定义的Web2认证方法登录他们的Web3账户。dApp可以验证来自Web2认证方法的签名，以此查询Web3账户地址，向Web2账户推送通知，并连接到Web3账户。用户不需要为不同的dApp创建其支持的不同的钱包。相反，dApp会自动连接到与用户Web2认证方式关联的Web3账户。用户账户和钱包不再是一体也不再受制于钱包，钱包本质上成为了用户可以登录账户来进行资产管理的dApp。

结论

EIP 4337将智能合约钱包及其相关基础设施标准化为六个合约接口：智能合约钱包主合约、打包器（Bundler）、入口点合约（Entry Point Contract）、代付合约（Paymaster）、钱包工厂（Wallet Factory）和签名聚合器（Signature Aggregator）。UserOperation是EIP 4337支持的新交易类型。

账户抽象市场有两类参与者：

具有社交登录、社交恢复、gas抽象、交易批处理，以及集成和聚合第三方服务（如法币出入金和DeFi协议）等功能的智能合约钱包。

基础设施（如Bundler或Paymaster）的第三方提供商，专注于模块化设计。

我对账户抽象市场有如下预测：

智能合约钱包市场竞争激烈，因为竞品的功能类似，技术壁垒也较低。

Bundler是一种盈利较难但是生态十分需要的公共物品。

账户抽象仍然是一个非常早期的市场，因为采用率仍然很低，EIP 4337尚未完成，许多L2尚未实现对账户抽象的支持。

账户抽象市场需要许多创新，如：无需许可的模块化基础设施、与现有法币和DeFi服务的集成、dApp SDK以及潜在的独立帐户层。