解读ERC7527：一种全新的去中心化定价模型

### ‍概述

如何对早期的想法或者应用进行定价，并参与投资？这在去中心化世界是一个巨大的挑战。ERC7527遵循“信贷创造货币”的共识，提出了一种基于Premium驱动的去中心化定价的范式。

在本文中，我们将深入介绍ERC7527的原理及协议细节，进一步介绍连续报价模型的运作原理及其巨大优势。同时，我们也会介绍一些基于ERC7527的可编程性和可组合性而产生的拓展应用。

### ERC7527

在上一篇文章中，我们介绍了目前常见的纯链上定价模型，其中也介绍了ERC7527的特殊的连续报价模型并给出了该模型的数学表示，但上一篇文章对于ERC7527的介绍是较为简单的，在本节中，我们将着重介绍ERC7527的原理、协议细节。

ERC7527提出了一种通过发行ERC7527通证给应用定价的新范式。ERC7527通证的价格实际上就量化了应用的市值(应用市值=ERC7527价格×数量)。这种方法将对应用的定价转化为了对ERC7527通证的定价。ERC7527通证的定价分为以下两个部分：

对于ERC7527通证的买入报价和定价的问题，ERC7527则充分提供了可编程性。具体来说，用户可以调用以下函数获取报价：



而ERC7527开发者则需要在开发过程中设计报价模型来最终给出报价。当用户调用getWrapOracle获得当前的报价后，用户可以选择向ERC7527的池子内注入流动性以实现报价到定价的转化。具体来说，用户可以调用FOAMM内的wrap函数来注入流动性：



对于ERC7527通证的卖出报价和定价问题，ERC7527使用了FOAMM构建的池子解决了卖出报价到定价的转化，用户可以通过池子将资产销毁、移除内部流动性实现退出。这里可能存在一个问题，即池子内的流动性由谁提供的问题。

在上文中，我们提到FOAMM通过wrap添加内部流动性，这部分就是池子内流动性的来源。ERC7527使用FOAMM将用户买入资产所支付的流动性注入到池子内以充当退出流动性。对于开发者而言，只需要给出卖出报价就可以，由于池子内存在流动性，卖出报价就是卖出定价。具体来说，开发者需要实现以下函数实现卖出报价功能：



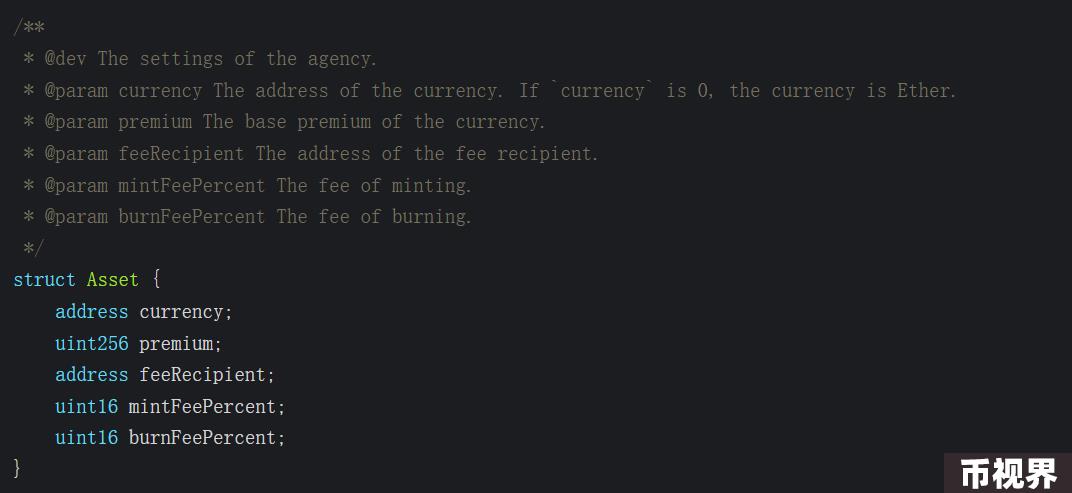
开发者可以基于池子的流动性构造任意的卖出报价函数，需要注意的是，卖出报价最终应当保持池子的清算平衡。用户可以随时调用getUnwrapOracle获取当前的ERC7527通证的卖出报价，并可以选择进一步调用FOAMM的unwrap函数来销毁ERC7527通证以获取池子内的卖出报价数量的通证：



总结来说，对于用户而言，ERC7527的交互仅有两部分：

在上文给出的getWrapOracle和getUnwrapOracle函数的返回值内都出现了price和fee选项，其中price代表当前wrap或者unwrap ERC7527通证所需要支付的通证数量，而fee则代表手续费，无论是wrap或unwrap操作，用户都需要支付一定数量的手续费，这些手续费最终被指定用户提取。

ERC7527是一种标准通证，与ERC20通证等类似，也有其通证地址。ERC7527存在如下定义字段：

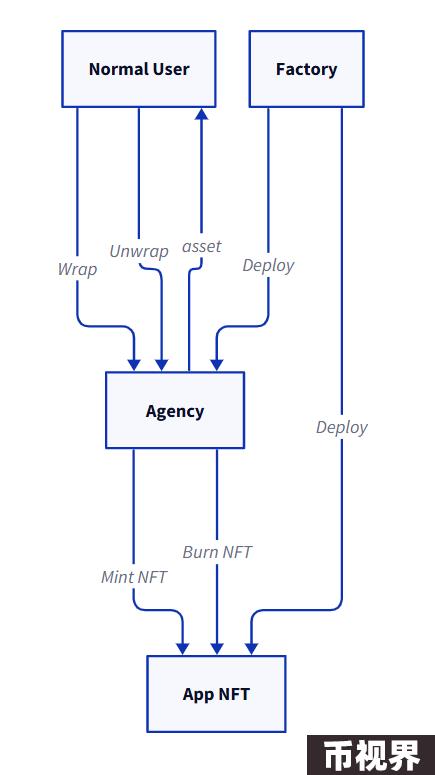


上述结构体定义了ERC7527最核心的属性，其中currency代表用户提供哪种通证来铸造ERC7527通证，premium指ERC7527通证的初始报价，feeRecipient代表wrap和unwrap交互过程中的手续费接收地址，而mintFeePercent和burnFeePercent则代表wrap和unwrap的手续费率。

ERC7527通证具有权利金(premium)的波动特征，在上文函数定义内使用了price一词，即Asset结构体内的premium。

ERC7527定义了三种不同的模块:

三种组件之间的关系可以参考下图：



ERC7527内的App和Agency是存在双向绑定关系的。从代码角度看，App的mint和burn函数仅能由Agency调用，而且ERC7527也约定了一系列函数以方便用户在已知App的情况查询Agency地址(getAgency函数)，和已知Agency的情况下查询App的地址(getStrategy函数)。

从流动性角度来看，App内的每一个ERC7527通证都有对应的池子内的的流动性，这保证了池子的清算平衡。

这种使用ERC7527通证对应用进行去中心化定价的方案是一种革命性的创新，这种方案引入了一种先发货币后建立应用的新范式。对于最早期的应用，开发者就可以直接使用ERC7527的Factory进行ERC7527通证部署。

用户可以调用wrap函数铸造ERC7527通证来参与项目的早期投资，也可以调用unwrap函数来退出投资。对于开发者而言，ERC7527通证的价格可以为早期应用提供有效的市值。随着应用开发和ERC7527通证持有者增加，应用市值增长，ERC7527通证的持有者和应用开发者都可以获得市值增长的激励。

### 连续报价模型

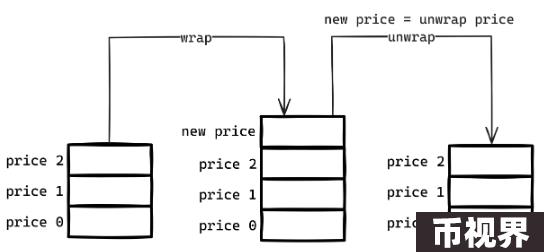
在上一节中，我们充分讨论ERC7527的运作原理，但上一节内并没有详细介绍买入报价模型的构造。ERC7527在此处仅提供了getWrapOracle接口，而报价模型具体实现完全依靠开发者。在本节中，本文将提出一种较为通用的买入报价模型。

ERC7527的买入报价应当同时考虑微观和宏观两种角度。

宏观来看，我们预期ERC7527通证的报价应当伴随着ERC7527通证持有量的增加而增加。更加具体来说，调用getWrapOracle获得的当前ERC7527通证的报价不应当低于定价(或称为“上一次的买入成交价”)。

ERC7527通证的报价应随着ERC7527通证持有量的增加而增加会带来一个很有趣的结果，即ERC7527通证本身将具有信用扩张机制。更加具体来说，会出现ERC7527通证市值大于TPL(Total Premium Locked)。这种信用扩张机制会使得每一个ERC7527通证的持有人获得信用的增长。

在上文，我们没有提及unwrap卖出交易，unwrap卖出交易会直接使得买入交易的报价下降。我们可以使用一种最简单的基于栈的系统来实现卖出推动买入报价下降的方案：



我们将ERC7527内所有的历史定价存储在栈内部，当每一次新的定价new price形成时，我们就将其推入栈内，买入报价模型会依赖于栈内最新的定价给出买入报价。

而当用户卖出时，我们直接将最新的买入定价new price作为卖出定价，从Agency内向用户转出流动性并同时将该定价从栈内弹出。此时报价模型还是依赖于栈内最新的定价price2对外给出报价。

由于price2是小于new price的，所以此时报价模型在unwrap后，对外给出的基于price2的报价也会低于过去以new price为基础给出的报价。当然，这种基于栈的卖出报价只是一种最简单的卖出报价方案。

从微观上，报价模型应该具有某种类似荷兰式拍卖的价格调整机制。荷兰式拍卖最大的特点是将时间引入了报价模型，随着长时间的无人成交，价格应当自动下降。

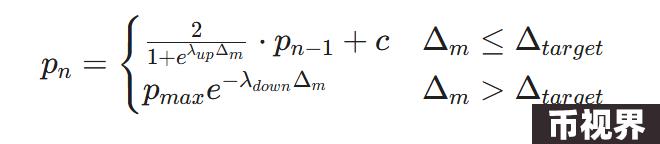
但荷兰式拍卖具有一个较为严重的缺点，由于随着时间推移价格一定会下降，存在等待博弈的囚徒困境，最终导致报价不断下降。一个良好的报价模型在微观层面上应当包含时间因子，且也应该存在上涨和下降两段。

综上所述，一个良好的报价模型应当符合以下要求:

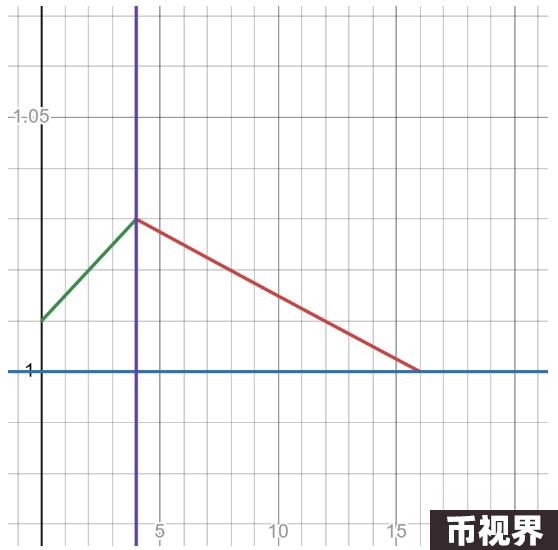
除此外，我们还希望报价模型可以满足以下应用的成长规律:

我们会在后文具体介绍解决方案。

首先，满足宏观要求实际上很简单，我们需要在报价模型内纳入定价（Pn-1）作为因子。然后，对于微观需求，我们可以将函数分段来解决问题。最终，我们可以获得以下报价函数：



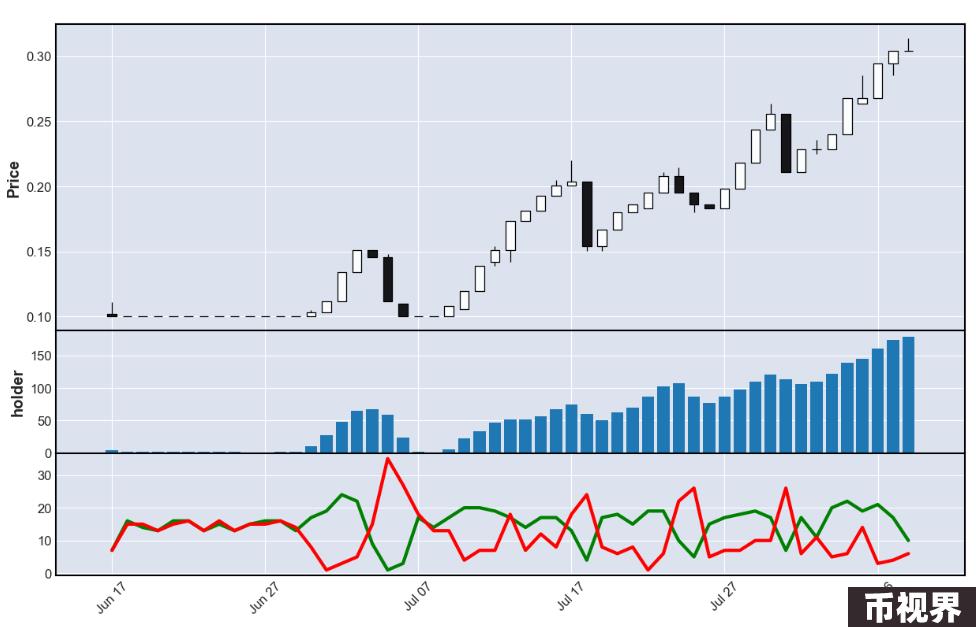
关于此报价函数的参数的具体含义，读者可以参考上一篇文章内给出的介绍。我们在此处直接给出此定价函数的图像：



在上图内，x轴代表当前时间距离上一次成交的时间间隔单位，而y轴代表当前报价与上一次成交价格之间的比值，值得注意的是，当价格下降到低于上一次成交价时，报价模型只会报出上一次等于上一次成交价的报价，这是为了满足报价模型的宏观要求。

我们直观看到上述报价模型既存在早期的上涨阶段，又存在后期的类似荷兰拍卖的下降阶段，这有效满足了报价模型的微观要求。此处，我们可以看到在上涨阶段的早期存在跳跃式上涨，这是为了避免在连续铸造过程中出现的ERC7527的持有量上涨但价格没有发生显著变化的情况。

我们建立一个环境来模拟上述报价模型，模拟结果如下：

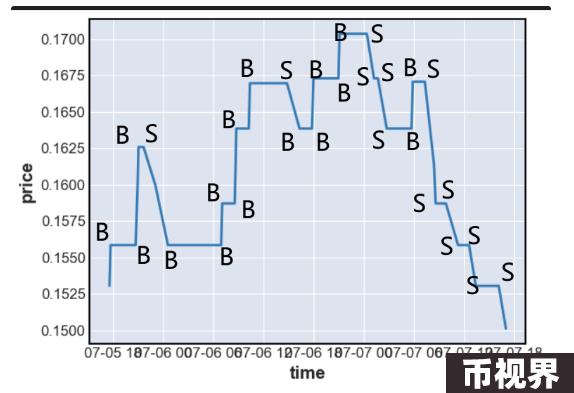


在模拟的走势K线图中，第一栏代表当前的ERC7527通证的价格走势，而第二栏则代表当前ERC7527通证的持有数量，此数量会随着wrap操作而增加，随着unwrap操作而减少。从代码层面来说，第二栏实际上是调用App的totalSupply函数获得的。第三栏则代表当前wrap和unwrap的次数，其中绿线则代表wrap的次数，而红线代表unwrap的次数。

上图展示了部分池子的早期运作情况，可能早期由于共识不足等原因，用户进进出出而导致定价长时间在初始价格左右波动，直到一些利好产生，用户选择买入但随后被早期用户砸盘返回初始价格，最后随着ERC7527通证持有人增加，定价在波动中震荡上行。

通过上述简单的模拟，我们可以看到基于连续报价模型的一个极其重要的优点，即用户的买入操作直接影响到了报价流程，这导致几乎没有可能两种ERC7527通证走出相同的K线图，极大提高了系统的博弈复杂度。

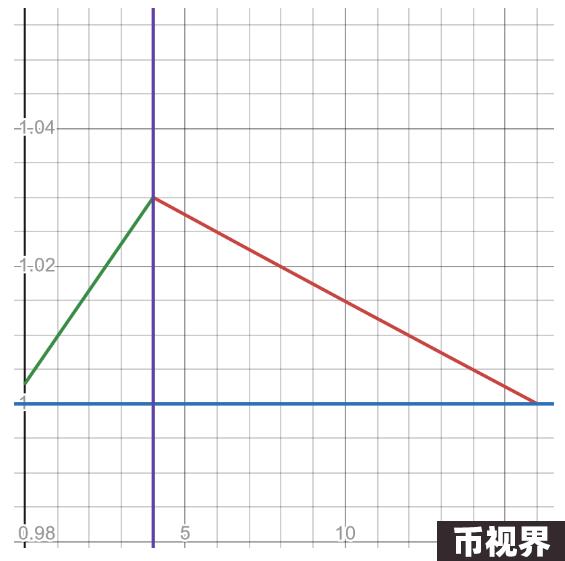
下图展示了一段具体的报价变化，其中B代表买入(wrap)，而S代表卖出(unwrap)。我们可以看到整个报价模型在宏观上每一次买入的成交价都不低于上一次的成交价，而每一次卖出都使用了之前已存在的买入定价，因此这个图像具有一定的对称性。



我们实际上解决了上述报价模型的宏观和微观要求，同时在报价上也满足了“早期阶段具有高成长性”的成长规律。接下来，我们需要解决一个问题，即如何在报价上实现应用后期的稳健成长性。早期报价的高涨幅是与应用的早期高成长相匹配。中后期的报价需要与应用的稳健发展相匹配，所以我们需要在中后期构建稳健成长的报价系统。

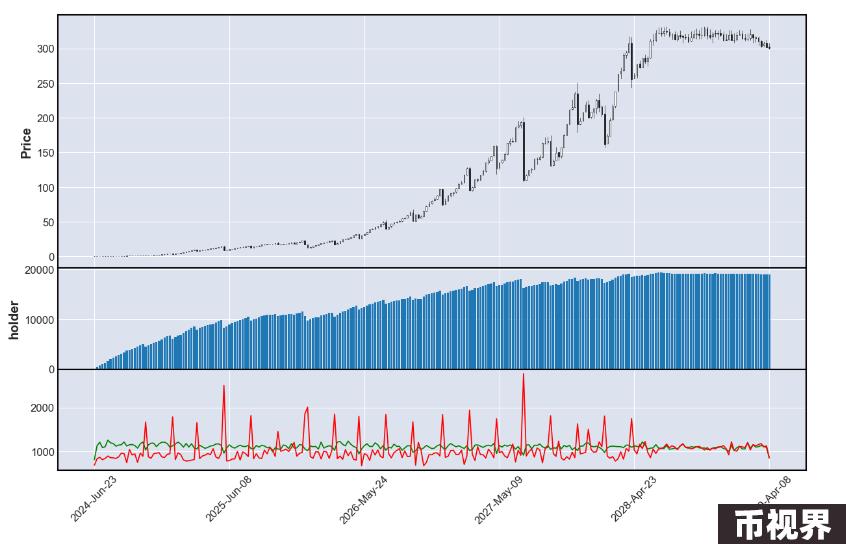
由此，我们决定通过分段解决此问题，即在早期报价内使用高成长性的报价系统，而在后期使用稳健成长的报价系统。当然，除了本文介绍的早期和后期的两段分割外，开发者可以根据自身情况进行更多段的分割，比如在早期、中期、后期使用不同的成长函数。

事实上，我们还是使用了与早期一致的报价函数，但修改了其部分参数，修改后的报价函数曲线如下：



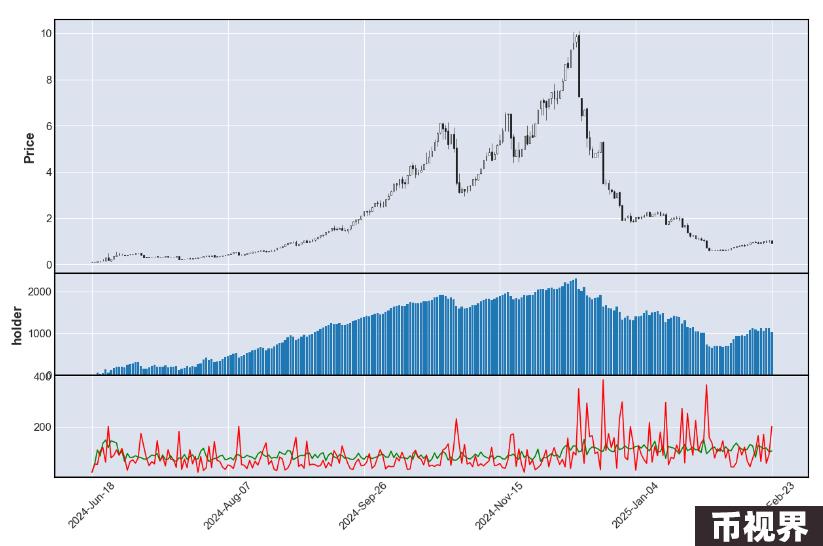
相比于早期报价函数，我们可以看到后期报价函数的初始涨幅更低。上图的x轴依旧使用了与上次成交间隔的时间单位作为x轴，但需要注意此处的时间单位是小于早期报价函数的。

我们尝试模拟ERC7527通证供应量从0达到20000期间的成交价格变化情况，我们获得了以下走势K线图：



上述走势K线图的设定的最初报价为0.1。在供应量达到20000时，价格从最初的0.1上涨为300，涨幅达到了3000倍。

当然，有些应用可能无法一帆风顺的走向成功。以下走势K线图展示了在早期蓬勃发展，突然遭遇危机导致用户退出的价格变化情况：



上述几个案例证明我们所提出的报价模型是有效的，其满足了应用的成长规律，既存在早期的高成长，也存在后期的稳健成长。这些优势是因为我们采用了分段的报价模型，在早期和后期使用了参数不同的报价函数。

另一方面，因为报价函数是连续的，所以用户可以等待报价函数给出预期的报价，而且报价的具体情况会随着用户的行为发生变化。这使得在本节介绍的报价模型内，不仅存在用户与报价模型的博弈，也存在的是用户之间博弈。而用户之间博弈是极其复杂，这导致很难根据报价模型反向推导出一种策略以获取无风险收益。

### 拓展应用

在本节中，我们将介绍ERC7527的以下几种拓展应用:

#### AI应用激励层

AI应用需要全新的激励层，而ERC7527可以为AI应用带来系统化的激励层，而ERC7527通证则是激励层的核心资产。

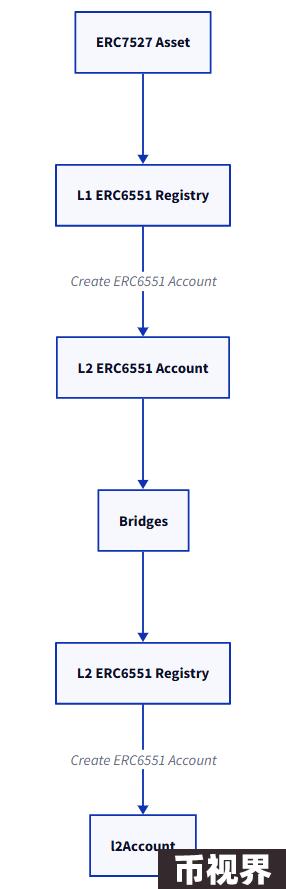
AI应用在发展的最早期部署ERC7527通证对其应用进行定价，此时AI应用就有了一个市场认可的估值，而用户也可以直接通过wrap操作来持有ERC7527通证投资AI应用。

从市值角度来看，伴随着AI应用的发展和ERC7527持有量的增加，AI应用的开发者和ERC7527通证的持有者都会获得市值上涨所带来的巨大激励。

AI应用也可以进一步使用分红措施激励用户。AI应用直接在以太坊二层要求用户使用通证进行支付。

用户可以调用合约来支付AI应用的费用。这些锁定在支付合约内的费用定期被划转到分红合约内，ERC7527通证的持有者可以直接在分红合约内提取收益，而AI应用开发者也可以在分红合约内部提取另一部分收益。

在技术实现上，ERC7527通证存在对ERC721的继承，这意味着ERC7527通证可以借助ERC6551协议生成ERC6551账户合约。而我们可以通过一些技术方案将ERC7527通证的ERC6551账户跨链到以太坊二层。具体技术实现路径如下：



当用户持有ERC7527通证后，可以调用位于以太坊主网内的ERC6551工厂合约来为ERC7527通证部署ERC6551账户，然后使用ERC6551账户调用部署在主网内的ERC6551桥接合约，桥合约会通过跨链桥调用位于以太坊二层内的ERC6551工厂合约为用户部署一个位于二层的ERC6551账户。

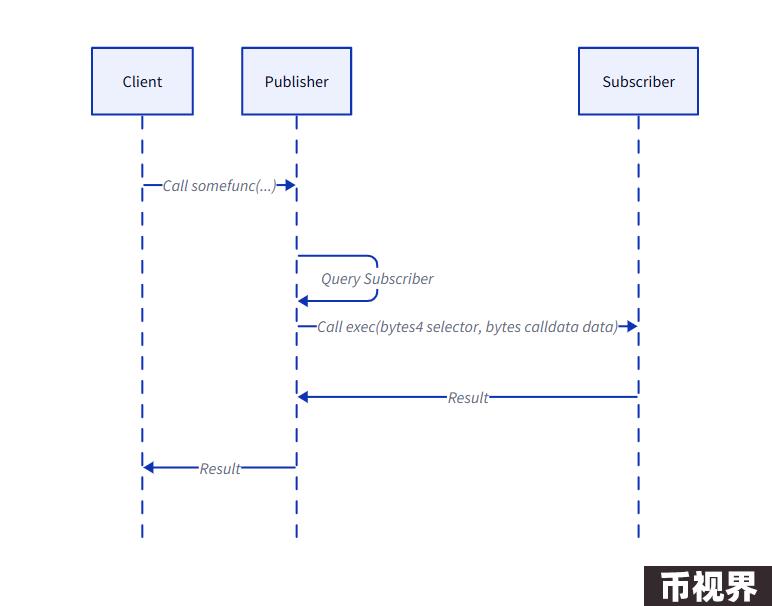
ERC6551跨链部分存在一个最简实现，开发者可以参考ERC6551Mirror内的代码，此仓库使用Chainlink CCIP实现了ERC6551的跨链操作。

当ERC6551跨链完成后，用户可以使用该ERC6551账户提取分红合约内的奖励。

#### 原子性预言机

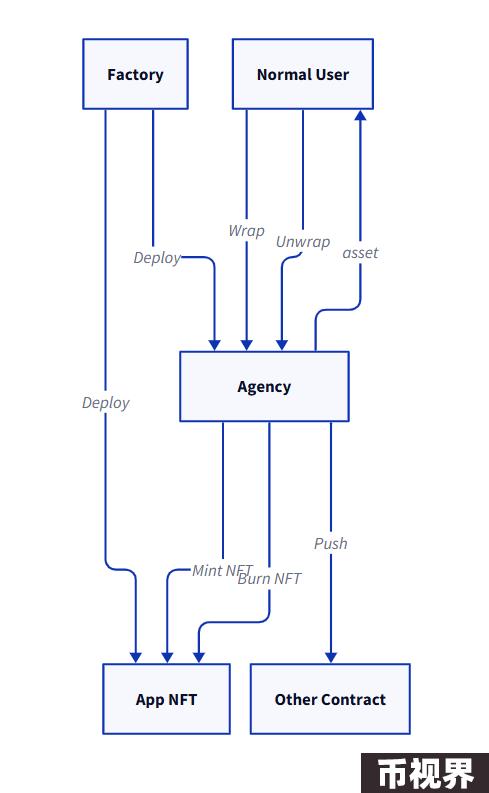
ERC7615是有趣的ERC，其具体内容是约定了一种原子性的通用的合约之间的推送关系。这种简单的推送协议可以构造出原子性的预言机，使得借贷协议等协议的开发难度显著降低。

ERC7615的运作原理如下图：



当用户(Client)调用发送合约(Publisher)的某些函数时，发送合约会查询该函数是否存在一些订阅合约，如果存在则调用订阅合约的exec函数来推送一些内容。上述流程最大的特点是原子性。假如订阅合约对于推送数据的处理出现问题，那么调用推送合约的交易(即上图内的Call somefunc(...)交易)也会直接报错。

在ERC7527内，我们多次提到ERC7527给出的卖出报价实际上就是卖出定价，任意持有ERC7527通证的用户都可以以卖出定价提取Agency内的流动性。那么，我们是否可以将ERC7527的卖出定价推送给借贷协议来实现借贷协议内的清算？



答案是可以的。而且由于连续报价模型在宏观上遵循买出报价随着ERC7527保有量上涨而上涨的属性，借贷协议并不需要关心ERC7527内的wrap操作，而只需要关心会导致价格下降的unwrap交易。

我们可以在unwrap函数内部嵌入ERC7615的推送，将当前的unwrap的卖出定价直接推送给借贷协议，当借贷协议获得推送的unwrap定价时，就可以根据其价格信息对借贷协议内使用ERC7527通证作为担保品的头寸进行清算。

一旦某一借贷头寸需要被清算，那么借贷协议可以执行unwrap操作提取ERC7527Agency内的流动性来平仓借贷头寸。

比较令人兴奋的是，上述操作是原子性的，即假如借贷协议的清算失败，那么ERC7527的unwrap操作也不会成功，这意味着借贷协议完全规避了穿仓风险。

事实上，任何协议都可以在ERC7527内获取其推送的卖出定价，这意味着开发者只需要编写一个ERC7615的接受接口，就可以获得一个资产的定价并进一步完成其他操作。

在ERC7527内，因为ERC7527通证在合约上存在对ERC721的继承，所以开发者可以对ERC7527通证进行其他赋能。当用户使用传统质押方案，比如直接将ERC7527转移给质押合约时，用户就丧失了ERC7527通证的其他权益，所以ERC7527通证需要一种全新的无需资产转移的质押方案。

基于ERC7615的推送机制，可以设计出这种无需资产转移的质押方案。具体来说，建立ERC7527Agency合约内的unwrap与质押合约的推送关系。质押的用户可以与ERC7527质押合约交互进行ERC7527通证的质押，但这一过程并不需要将ERC7527通证转移进质押合约。

此时，ERC7527通证可以获得质押合约内分红。当用户unwrap已质押的ERC7527通证时，Agency合约会使用ERC7615将此消息推送给质押合约，质押合约将解除指定ERC7527通证的质押状态并进行最终清算。

在此过程中，ERC7615保证了质押系统的清算问题，确保了已销毁的ERC7527通证不可能获得质押收益。