预测市场白皮书

PickYesNo 是一个基于 Polygon PoS 区块链的去中心化开源预测平台，致力于在任何情况下确保用户资产的绝对安全，包括平台遭受黑客攻击、内部监守自盗或服务终止等极端场景。

利用 Polygon PoS 的公平性、透明性和不可篡改性，PickYesNo 通过核心智能合约：钱包合约、预测合约、预言机合约一同构建了一个稳健且值得信赖的资产管理与交互框架。PickYesNo 优先考虑用户信任与资产安全，即使需要承担较高的 gas 费用，也坚持实施严格的安全措施。

本白皮书详细阐述了 PickYesNo 的技术架构和安全机制，展示了其在面对威胁时的韧性以及对用户资产保护的承诺。

术语

智能合约：部署在区块链上的自执行程序，根据预定义规则管理用户与平台之间的交互，确保透明性和不可篡改性。

钱包合约：核心智能合约，负责安全管理用户资产，并通过相应机制防止未经授权的访问或资产损失。

预测合约：支持用户对特定结果进行交易的合约，确保基于正确结果的结算和利润分配。

Chainlink 预言机合约：基于 Chainlink 链上价格数据，专门用来为加密币预测提供结果。

预言机合约：将外部数据引入区块链的合约，为除了加密币以外的预测合约提供可靠且防篡改的现实世界事件结果。

工厂合约：用于生成钱包合约、预测合约的智能合约，确保所有合约代码安全一致。

EOA：外部拥有账户（Externally Owned Account），由私钥控制的区块链账户，用于绑定钱包合约以确保资产安全。

Gas 费用：在区块链上执行交易或智能合约操作所需的计算成本，PickYesNo 在优先保障安全的同时优化 gas 消耗。平台会承担几乎所有 gas 费用。

用户资产：用户在平台上持有的 USDC，受智能合约保护，免于损失或未经授权的访问。

黑客攻击：外部攻击者未经授权入侵平台基础设施或智能合约，试图窃取或操纵用户资产。

监守自盗：内部人员的不诚信行为，试图挪用用户资产，平台通过透明和去中心化的设计加以防范。

服务终止：平台停止运营的情况，PickYesNo 的智能合约确保用户仍能完全控制和访问其资产。

区块链不可篡改性：区块链数据无法更改的特性，确保平台交易历史和合约逻辑的完整性。

引言

本平台通过区块链技术确保用户资产的绝对安全。为此，PickYesNo 精心设计了智能合约（Smart Contract），使用户的资产在各种极端情况下都不会有丝毫损失。这些极端情况包括但不限于：

1. **平台被黑客入侵**

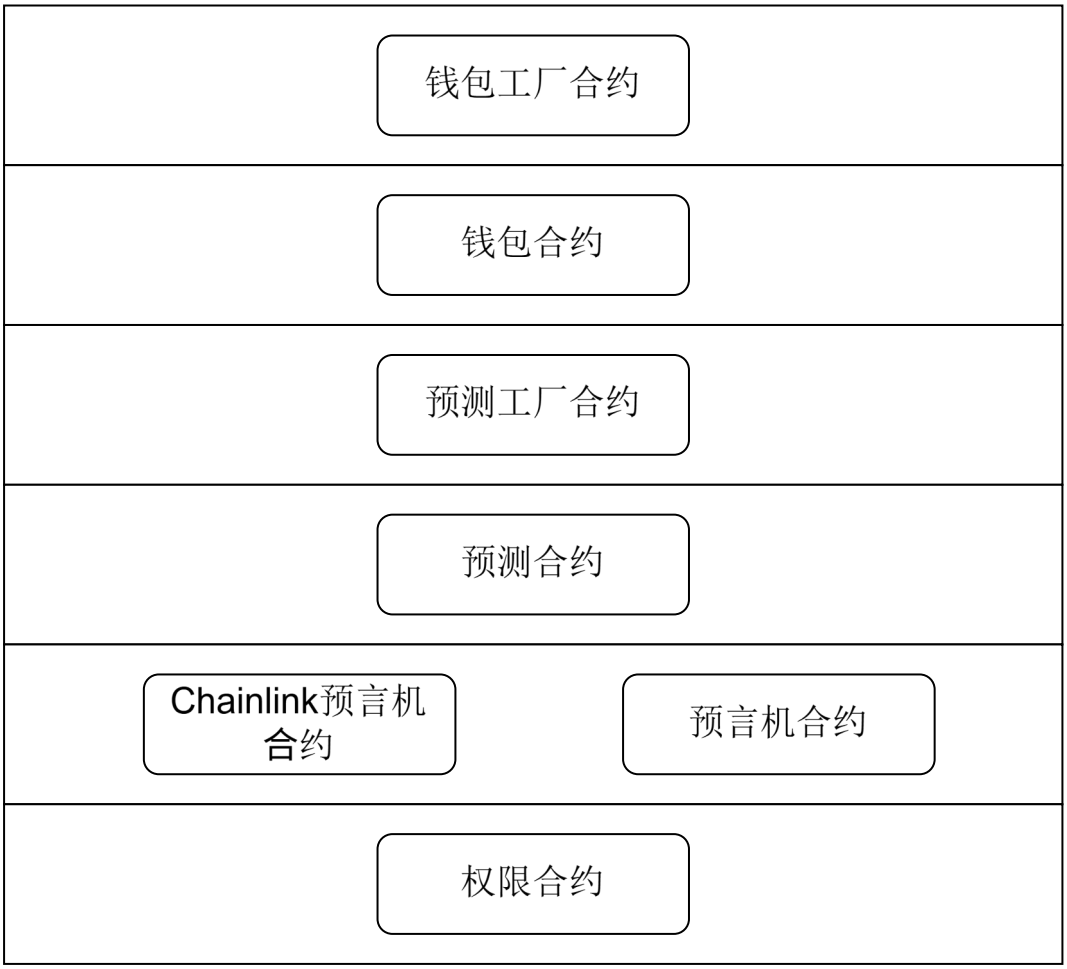
2. **平台监守自盗**

3. **平台终止服务**

我们并不是为了「上链而上链」，PickYesNo 真正发挥了区块链公平、公开、不可篡改的特性，来保障用户的资产安全，甚至不惜承担高昂的 gas 费用，只为提供一个值得信赖的平台。

下面，让我们从技术视角深入剖析 PickYesNo 的智能合约，验证平台是否真正做到了上述的安全保障。

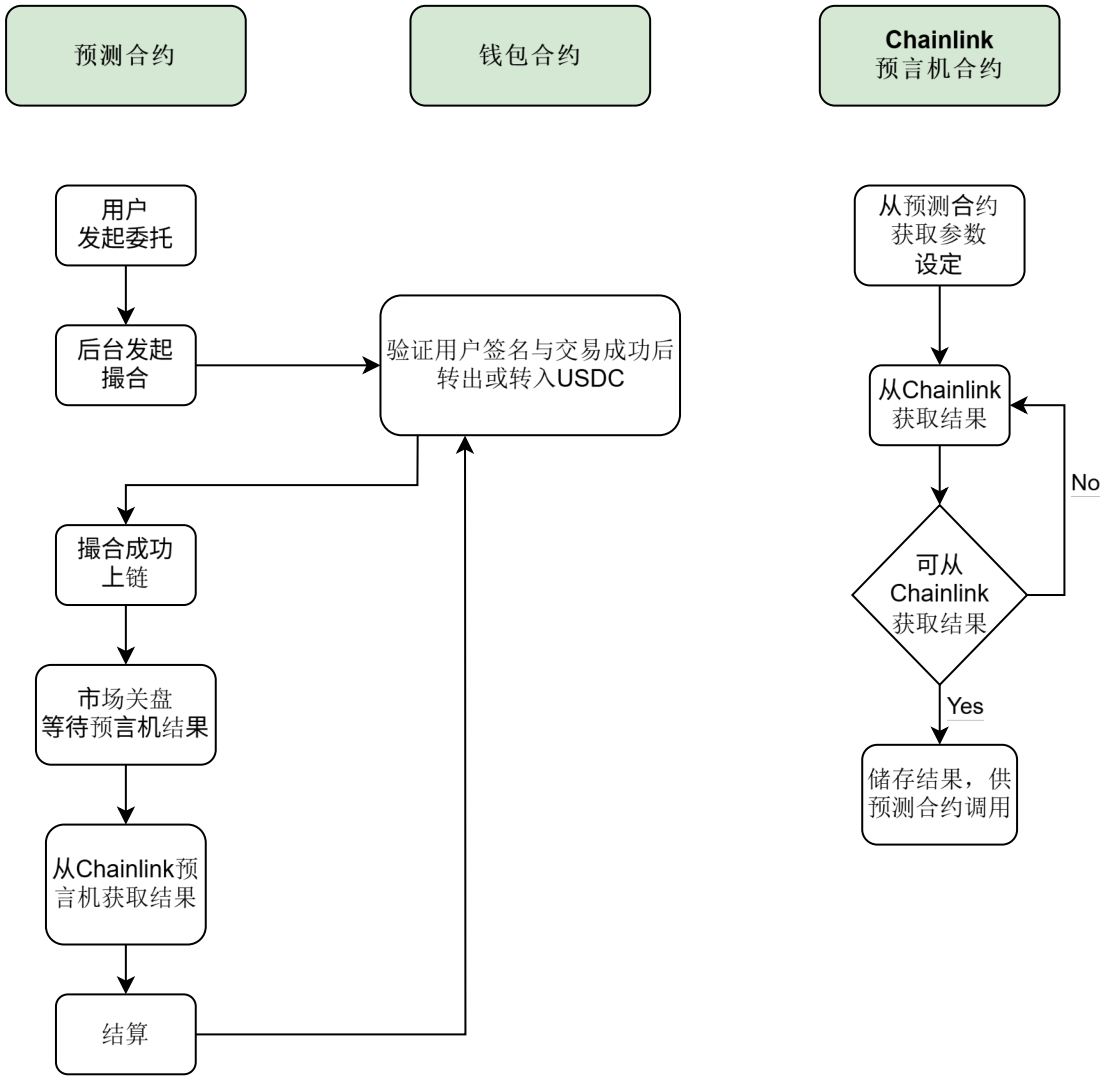
PickYesNo 的智能合约概览如下图，其中最重要的合约是：**钱包合约、预测合约、预言机合约。**

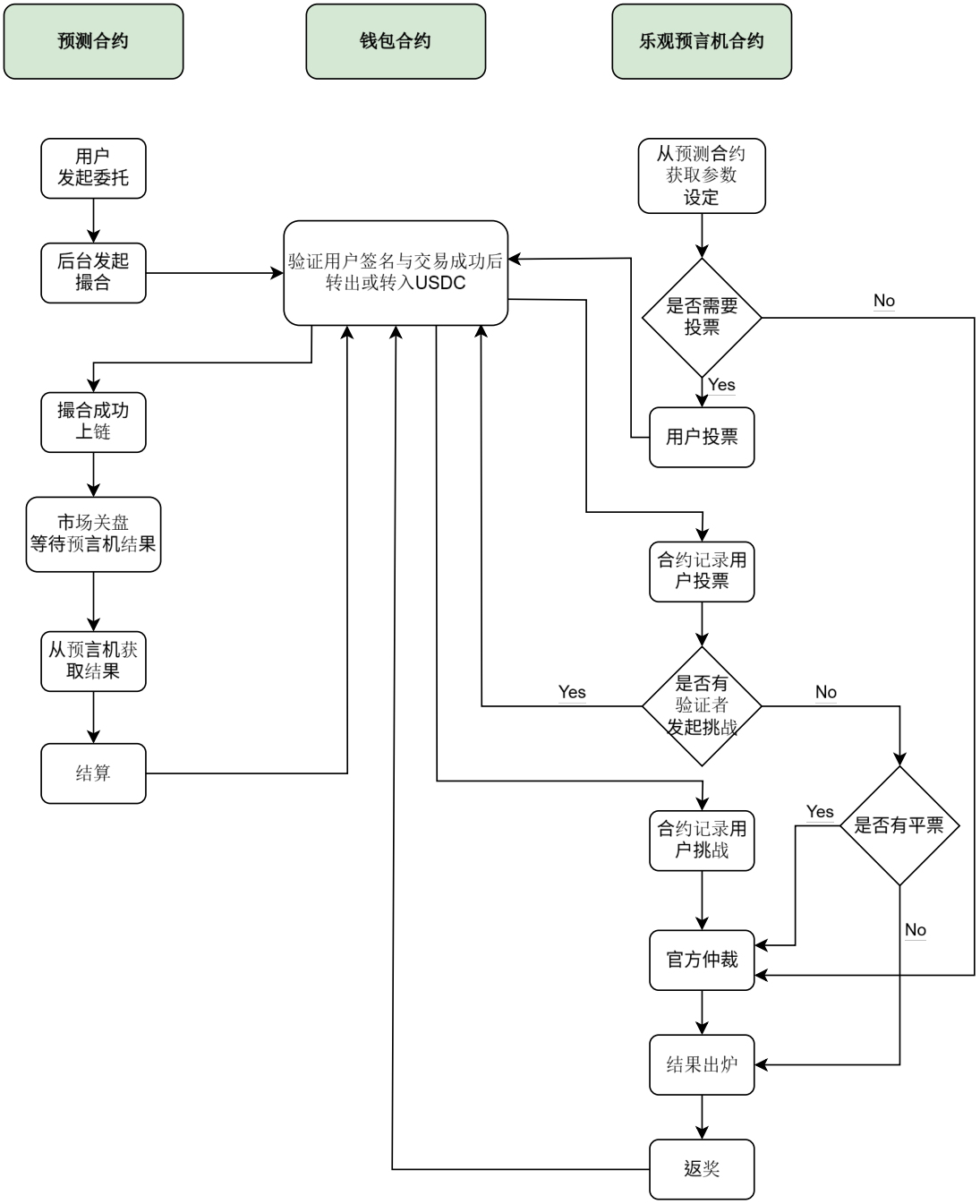


整体流程概览

以下是 PickYesNo 平台从预测市场建立、委托、撮合、预言机开出结果并结算的运作概览。

加密币预测市场和一般预测市场使用的预言机不同，前者用 Chainlink 预言机，后者使用基于投票方式的乐观预言机。预测合约和钱包合约则完全相同。这些合约在各自的章节中有更详细的描述。




预测合约
钱包合约
乐观预言机合约
用户
发起委托
后台发起
撮合
撮合成功
上链
市场关盘
等待预言机结果
从预言机获
取结果
结算
验证用户签名与交易成功后
转出或转入USDC
从预测合约
获取参数
设定
是否需要
投票
No
Yes
用户投票
合约记录用
户投票
是否有
验证者
发起挑战
Yes
No
合约记录用
户挑战
Yes
是否有平票
No
官方仲裁
结果出炉
返奖

钱包工厂合约

在介绍钱包合约之前，必须先了解我们使用的钱包工厂合约。每当调用它时，它会在新的地址部署一个钱包合约，所有的钱包合约均由它生成，代码完全一致，安全性相同。我们只有在新用户创建账户时才会调用它。

如果您发现您的钱包合约不是由官方钱包工厂合约生成的，那肯定出现了问题！此时请立即停止与该钱包合约交互，并尽快与我们联系。

官方的钱包工厂合约地址请查看：https://pickyesno.com/docs/contract/contract-address，请务必确认您的钱包工厂合约来源于官网所示的地址

钱包合约

PickYesNo 强烈建议用户在注册时立即将分配的钱包合约绑定自己控制私钥的 EOA，只有这样，才能享受平台提供的最高等级安全保护。具体绑定方式请参考**《绑定自己的 EOA》**。

PickYesNo 希望为推动 Web3 应用的普及尽一份力，让更多没有接触过区块链的用户也能轻松上手，享受 Web3 的便利。因此我们精心设计了钱包合约，让用户在没有 POL 的情况下，也能安全的与链上交互，即使平台需要额外承担更多的 gas 费用。

绑定钱包(bindWallet)

为确保您的资产安全，绑定钱包是最关键的一步。绑定钱包就是连结钱包合约与您自己可以控制私钥的 EOA 的过程。绑定之后，钱包合约就在您的 EOA 控制之下。

如果您不关心技术细节，只需在注册账户时，按照以下步骤操作，就能确保您的资产 100%安全：

1. 确认您即将绑定的钱包合约，是由官方的钱包工厂合约生成的。如需了解验证方法，请参考《**钱包工厂合约**》。
2. 点击[绑定钱包]功能，按弹出框的提示链接您拥有私钥的 EOA 钱包。
3. 确认钱包合约发出的 WalletBound 事件的第三个参数，与您输入的 EOA 一致。您可以按如下方式确认事件：
    - 打开官网 https://polygonscan.com
    - 通过钱包合约地址查询出 transaction hash
    - 通过 transaction hash 查询出 Logs
    - 确认 WalletBound 事件的第三个参数，是否和您绑定的 EOA 一致
4. 绑定成功后，除了您本人之外没有任何人能够转移您的资产。

在绑定钱包后，所有将资产转出的操作，都需要您的签名(遵循 EIP712 规范)，只有确认是您本人后，才能正常执行。只要签名不对，整笔交易就会回滚。绑定钱包后，不管是平台还是黑客，都无法将资产从您的钱包偷走，只有您本人才动用里面的资产。

参考代码:

function name: bindWallet

input:

1. uint256 requestId，用于服务器识别本次交易 ID
2. address wallet，用户希望绑定的 EOA 地址
3. bytes signature，绑定 EOA 的签名

output:
无返回值

绑定 EOA 的函数是 bindWallet。第一次调用必须由平台执行，gas 费用由平台承担。一旦绑定成功，只有绑定的 EOA 可以调用。平台如要再次调用，则需要绑定 EOA 的签名。

在上述的第 1 步中，验证钱包合约是否来自官方的钱包工厂合约，目的是防止后台被黑客攻击，或内部人员恶意提供假冒钱包合约。由于区块链数据不可篡改，这是最保险的做法。

基于同样的原因，第 3 步要求确认链上事件，防止黑客或内部人员动手脚。通过链上 Logs 验证，您可以确保合约实际绑定的 EOA 与输入一致。

重新绑定

如果您在钱包合约成功绑定 EOA 后，想更换绑定到另一个 EOA，您需要使用已绑定成功的 EOA 账户主动调用钱包合约中的 bindWallet 函数，或是提供签名给平台，平台才有权限调用，并代您支付手续费。

参考代码：

function name: bindWallet

input:
1. uint256 requestId，服务器用于识别本次交易 ID。如果您是直接与智能合约交互（未通过平台），可以随意输入任意的 uint256，不会对您的账户安全造成任何影响。
2. address wallet，想要重新绑定的新 EOA 地址。
3. bytes signature，绑定 EOA 的签名

output:
无返回值

扣款

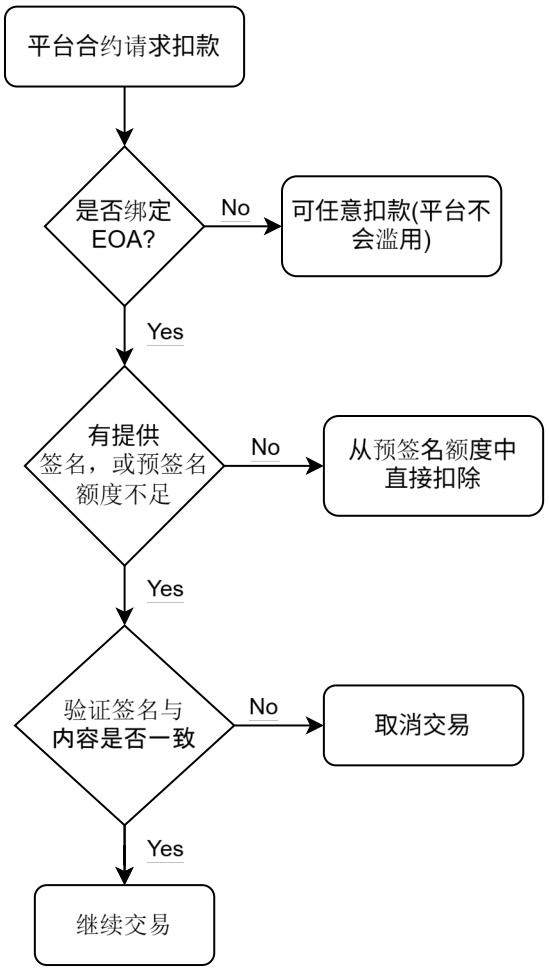
在您使用平台的服务时，有两种类型的智能合约可以从钱包合约中扣款：预测合约以及预言机合约。

- 在未绑定 EOA 时，这两类智能合约可以随时从钱包合约中扣款。
- 一旦绑定了 EOA，任何人，包括平台的智能合约，都无法在未经您本人同意的情况下动用您钱包合约内的资产。

在绑定了 EOA 的情况下：

- 如果有提供签名，或是预签名额度不足，就会检查您的签名内容是否与本次交易信息一致。
- 反之，如果没有提供签名，且预签名额度足够，就会使用预签名额度。。

扣款的流程图可以概括为：



不同类型的合约（预测合约、预言机合约）在进行扣款时，签名内容（encodedData）结构各有不同，具体可以在各自章节中查看详情。

验证签名与内容一致性的流程（适用于所有合约）如下：

1. 从签名（signature）中解析出 r、s、v。
2. PickYesNo 遵循 EIP-712，这个协议能还原出发送者地址。
3. 比对还原的地址，是否和您绑定的 EOA 相同。相同则继续交易，否则交易回滚。

这种机制与以太坊标准交易验证方法一致：通过签名和数据还原出发送者钱包地址，确保交易确实由您本人授权。

托管(entrust)

您可以在不支付链上 gas 费的情况下，将 USDC 转移到钱包合约。只要您提供遵循 EIP2612 的 r、s、v 签名，平台就会替您转移 USDC，并负担全部的链上 gas 费。

参考代码：

function name: entrust

input:
1. uint256 requestId，用于服务器识别本次交易 ID。
2. address from，您的 USDC 将从 from 地址转移到钱包合约。
3. uint256 amount，要转入钱包的 USDC 数量。需要注意的是，USDC 的 decimal 是 6。因此如果您直接与合约交互，比如输入 50000（$5*10^4$），实际转账的是 0.05 USDC；如果您要转 3 USDC，需要输入 3000000（$3*10^6$）。
4. uint256 deadline, EIP2612 中，签名有效的截止期限。为 unix timestamp。
5. uint8 v, EIP2612 签名中的 v。
6. bytes32 r, EIP2612 签名中的 r。
7. bytes32 s, EIP2612 签名中的 s。
output:
无返回值

赎回(redeem)

redeem 函数能将钱包合约内的资产转移至指定地址。

调用 redeem 函数的权限，仅限绑定的 EOA 和平台系统。可能您会担心：既然平台也能调用，会不会存在安全隐患？下面我们分别针对两种情况说明：

1. **尚未绑定 EOA**

   理论上，平台可以将您钱包合约中的资产转移到任意地址。虽然平台绝对不会这么做，但这的确存在一定安全风险。

2. **已绑定 EOA**

   将钱包合约绑定 EOA 能完全保证您资产的安全。因为：

   1. 若是由平台赎回，且转帐目标不是绑定 EOA，会需要您的签名。没有您的签名，平台只能将赎回金额转至您绑定的 EOA。
   2. 若是由您绑定的 EOA 赎回，不用签名，因为可以确定是您本人的操作。

   通过将钱包合约与 EOA 绑定，您的资产将受到严格保护，只有您授权的交易才能执行，确保资产安全无忧。

参考代码：

function name: redeem

input:
1. uint256 requestId，服务器用于识别本次交易的 ID。如果您是直接与智能合约交互（未通过平台），可以随意输入任意的 uint256，不会对您的账户安全造成任何影响。
2. address wallet，USDC 要转出的目标地址。如果您已绑定 EOA，输入的地址必须等于您绑定的 EOA，或签名过的地址，否则函数会直接 revert。
3. uint256 amount，要转出的 USDC 数量。需要注意的是，USDC 的 decimal 是 6。因此如果您直接与合约交互，比如输入 50000（$5*10^4$），实际转账的是 0.05 USDC；如果您要转 3 USDC，需要输入 3000000（$3*10^6$）。
4. bytes signature，签名。用来验证交易是否来自绑定 EOA。只有在绑定 EOA 后，且转出地址与绑定 EOA 不同时，才会用到。

output:
无返回值

营销奖金(present)

我们先来定义资产与营销奖金。营销奖金只能用来与预测市场或预言机合约互动，而资产还能让您赎回，即从钱包合约内转出。

在一些特殊活动，例如春节、周年庆时，平台可能会发放营销奖金到用户钱包。您可以使用这笔钱进行预测交易或预言机投票。而您在这些活动中获利后，营销奖金就会转为您的资产，让您可以随意动用；反之，若是您在这些活动中失利了，您的资产也不会有任何减少，只是可用的营销奖金减少而已。

营销奖金有严格的的安全机制，如果平台想要增加营销奖金，就一定要转入等值的 USDC 到您的钱包，这样可以防止平台恶意增加营销奖金，导致您能赎回的资产减少。在不转 USDC 到您的钱包的情况下，平台只能减少营销奖金的值，或是在减少营销奖金的同时，转走等量的 USDC。您的资产不会因此减少，因为平台只能转走之前转来的营销奖金。

营销奖金的机制只有可能让您的资产增加(您用营销奖金获利时)，不可能减少您的资产。

参考代码:

function name: present

input:
1. uint256 requestId，服务器用于识别本次交易的 ID。
2. uint64 amount，增加的营销奖金的值。一定会有等量的 USDC 转进您的钱包，否则交易回滚。
3. address from，USDC 将从这个地址转到您的钱包。
4. bytes32 code，授权执行者 EOA 将营销奖金转入您的钱包的营销代码。
5. bytes signature，授权执行者 EOA 将营销奖金转入您的钱包的授权签名。

6. uint64 unlocked, 当您用营销奖金获利时，平台会用这个参数，将您的获利转为您的资产。
7. uint64 reclaimed, 平台能用这个参数收回营销奖金。只会减少营销奖金，绝对不会减少您的资产。

预签名(preSign)

如果您对平台已经有了较高的信任度，同时希望简化操作，避免每次交易都手动签名，您可以使用预签名功能。

通过预签名，您将授权平台一定额度，可以在委托单撮合成功时或投票时直接将您的 USDC 质押到相应合约中，无需每次都手动签名。额度使用完后，需要重新提交新的预签名。您也可以透过把预签名金额设为 0，来取消预签名的额度。

需要特别说明的是，预签名额度通常会消耗得比实际动用的 USDC 还要多。例如您在预测市场提交市价单，而实际成交价是 50 美分时，只会移转 50 美分的 USDC，但会直接扣除您 99.9 美分的预签名额度。

您可以放心，这样完全不会导致您有任何资产上的损失，因为实际移转的金额会完全等于实际成交价，只是预签名的额度会消耗比较快。之所以这样做，是因为在您提交市价单时，平台无法得知实际成交金额，因此会先扣掉预测市场的最大成交价，也就是 99.9 美分，以避免预签名额度不足，导致交易回滚。

参考代码：

function name: preSign

input:
1. uint256 requestId，服务器用于识别本次交易 ID。

2. uint64 amount，您愿意授权的 USDC 数量。可以设定超过钱包当前余额的金额，但实际扣款时余额不足仍会导致交易失败（revert）。
3. bytes signature，您提供的离线签名，用于验证操作是经过您本人同意的。每个签名都会包含不同的 nonce，从而防止重放攻击（replay attack）。

output:
无返回值

在钱包合约未绑定 EOA 时，预签名 preSign 的函数无法被调用。而绑定 EOA 后，仅有绑定的 EOA 和平台可以调用。

不管是谁调用预签名函数，都需要提供绑定 EOA 的有效签名，避免平台滥用权限，随意增加用户的预签名额度。为了防止重放攻击，平台会将 nonce 和请求内容一同进行签名处理。每次预签名成功后，钱包合约内部的 nonce 都会自增。

注意，预签名的授权金额可以大于钱包合约当前持有的 USDC 数量，但在实际使用时，如果余额不足，交易仍会失败（revert）。

救援 ERC-20 代币(rescueToken)

如果您不小心将 USDC 以外的 ERC20 代币转入钱包合约，平台有提供救援代币的函数，让您可以将代币转出来。

出于安全考量，并非所有 ERC20 代币都支持救援，只有合适的代币会被加入允许救援的白名单，平台保留随时修改白名单的权力。

仅平台可以调用这个函数。

参考代码：

function name: rescueToken

input:
1. uint256 requestId，用于服务器识别本次交易 ID
2. address from，ERC20 代币的合约地址。
3. address to，将代币转移到 to 地址。

output:
无返回值

回收(recyleWallet)

万一您忘记了帐号密码或私钥，导致无法操作尚有余额的钱包合约，平台可以在您提供所有权证明后，将合约内的资产取出，并退还给您。

若钱包合约不符合回收条件，即使平台执行 recycleWallet 函数，您的钱包也无法被回收，资产无法被改动。对于符合条件的钱包，在未接到遗忘帐户密码申诉时，即使平台有执行的权力，也不会回收。

对于活跃用户，您无需担心钱包合约会被无故回收，因为回收的前提就是钱包已经长期没有托管或交易记录。具体的回收规则为：

- 如果钱包合约内有 USDC，
  - 而且有托管或交易纪录，需要距离上次托管或交易超过一年才可回收。
  - 而且无托管或交易纪录，需要在本次调用 recycleWallet 函数一年内，仍然没有托管或交易记录才可回收。
- 如果钱包合约内没有 USDC，
  - 而且有托管或交易纪录，需要距离上次托管或交易超过 180 天才可回收。

- 而且无托管或交易纪录，需要在本次调用 recycleWallet 函数 3 天内，仍然没有托管或交易记录才可回收。

参考代码：

function name: recycleWallet

input:
1. uint256 requestId，用于服务器识别本次交易 ID。

output:
无返回值

升级钱包(upgradeWallet)

本函数包含三种功能：升级钱包合约本身的逻辑、升级预测工厂合约，及升级预言机合约。

1. 升级钱包合约本身的逻辑

PickYesNo 的钱包合约使用代理合约(ERC1967)的模式。倘若之后钱包推出新功能，用户可以自行决定是否要升级其运行逻辑。

升级逻辑需要用户签名，您可以先在链上确认开源的升级内容，再决定是否提供签名。

因为需要签名，即使是平台执行者 EOA 私钥泄漏，只要没有您的签名，黑客也无法将您的钱包升级为恶意逻辑，绝对保障您的资产安全。

2. 添加预测工厂合约

预测合约能从钱包合约扣款(详见扣款章节)。那么，钱包合约是如何确认某个预测合约是否合法呢？

钱包合约不直接记录预测合约是否合法，因为预测合约太多了。钱包合约内只会记录预测工厂合约地址。预测合约由工厂生成时，其地址会被记录至权限合约中。钱包合约在扣款前，会确认该预测合约是否是由您认可的预测工厂合约生成。只有经您认可的工厂合约生成的预测合约，才能从钱包合约扣款。

钱包合约生成时，会内建一个预测工厂合约。藉由 upgradeWallet 函数，您能添加预测工厂合约到您的授权名单。链上会有新工厂合约的代码供您检视。

添加工厂合约需要您的签名，以及权限合约的认证，这让平台或黑客均无法任意修改您的授权名单，完全确保资产安全。

3. 添加预言机合约

在您授权后，预言机合约能从钱包合约扣款(详见扣款章节)。

与预测合约不同，同一时间只会有一份预言机在运作，因此没有采用工厂合约的形式。钱包合约会直接记录预言机的地址。藉由 upgradeWallet 函数，您的钱包将认可新的预言机合约。链上会有新合约的代码供您检视。

添加预言机合约同样需要您的签名，这让平台或黑客均无法任意修改您的授权名单，完全确保资产安全。

参考代码:

function name: upgradeWallet

input:
1. uint256 requestId，服务器用于识别本次交易的 ID。
2. address newImplementation, 新的钱包合约逻辑地址。倘若本次不更新钱包逻辑，输入 address(0)。
3. address newPredictionFactory, 欲升级的预测工厂合约地址。倘若本次不更新工厂，输入 address(0)。
4. address newOracle, 欲升级的预言机合约地址。倘若本次不更新预言机，输入 address(0)。5. bytes signature, 用户签名。只要有绑定 EOA，不管是升级钱包逻辑、预测工厂还是预言机，都需要提供签名。

预测工厂合约

在详细介绍预测合约前，必须先了解预测工厂合约：

1. 每当调用预测工厂合约时，它会在新的地址上部署一份新的预测合约。所有的预测合约由预测工厂合约生成，代码完全一致，安全性一致。

2. 每当有新的预测市场需要创建时，平台都会调用预测工厂合约。建议您在交易前，务必确认互动的预测合约是由官方预测工厂合约部署，以确保资金安全。

官方的预测工厂合约地址请查看：
https://pickyesno.com/docs/contract/contract-address，请务必确认您的预测工厂合约来源于官网所示的地址

预测合约

我们的预测合约代码支援多种类型的预测市场，您能透过函数 getSetting(详见下方章节)回传的结果，判断当前预测市场的类型。

不同预测合约使用的预言机合约可能不同，但都能给出完全符合事实的结果。您可以在后面的章节了解不同预言机如何运作。

每个预测选项只有**是**或**否**。如果一个市场有 N 个选项，就会有 N 个「是」和 N 个「否」供选择。用户在交易时，不需要对所有选项都做选择，可以只选择

自己认为期望收益最高的。甚至可以在同一选项上选择“是”和“否”，一切依您的策略而定。

我们祝愿那些开发出优秀策略的用户能够获得更多收益！

预测合约的结果类型

预测合约的结果类型分为两种：

- **单一结果**

  在同一轮中，不同选项互斥，仅有一个结果会发生。

  举例：2049 年 X 国总统大选，有 A、B、C 三位候选人，最终只能有一人当选，不可能出现多人同时胜选的情况。

- **独立结果**

  在同一轮中，不同选项可以同时发生，互不排斥。

  举例：2049 年 X 国大选，Y 党候选人在 A、B、C 三个州的胜负情况。候选人可以在零个、一个、两个或三个州获胜。

交易撮合

交易流程如下：

1. 用户通过平台委托
2. 后台匹配对手单
3. 后台调用 broker 方法，上链撮合结果，订单正式成立，资产转移

我们来验证交易的四大安全机制：

1. **撮合未成功时，平台是否能动用您的资产？**

   不能！只有当撮合成功时,broker 才能正确执行。如果平台匹配了不适当的吃单方和挂单方,链上的交易会 revert。

2. **撮合成功后，链上数据是否与交易数据一致？**

   是的，且即使有差异，也必须是对用户有利的成交价格。

3. **撮合与资产转移时，金额是否正确？**

   是的，转移金额严格根据委托单及用户签名信息计算。

4. **资产转移后是否仍安全？**

   是的，资产保管在预测合约中，安全无虞。

需要特别说明的是，假设您在预测市场提交市价单，而实际成交价是 50 美分时，平台只会移转 50 美分的 USDC，但会直接扣除您 99.9 美分的预签名额度。

您可以放心，这样完全不会导致您有任何资产上的损失，因为实际移转的金额会完全等于实际成交价，只是预签名的额度会消耗比较快。

之所以这样做，是因为在您提交市价单时，平台无法得知实际成交金额，因此会先扣掉预测市场的最大成交价，也就是 99.9 美分，以避免预签名额度不足，导致交易回滚。

broker 方法詳解

参考代码:

function name: broker

input:
1. uint256 requestId，服务器用以识别交易 ID
2. uint256 optionId，委托的选项 ID

3. Order taker，吃单方的委托单信息
4. Order[] maker，挂单方的委托单信息数组

output:
无返回值

Order 结构定义

- uint256 mode：1=市价单，2=限价单
- uint256 buysell：1=买是，2=买否，3=卖是，4=卖否
- uint256 expectedPrice：当 mode=1，代表市价单期望的委托总额（6 位小数，单位 USDC）；当 mode=2，代表限价单期望的买卖价格（6 位小数，单位 USDC）
- uint256 expectedShares：当 mode=1，代表市价单实际成交的委托总额（6 位小数，单位 USDC）；当 mode=2，代表限价单期望的股份数（6 位小数）
- uint256 matchedPrice：实际成交价格（6 位小数，单位 USDC）
- uint256 matchedShares：实际成交股份数（6 位小数）
- uint256 fee：手续费率（单位：万分之一）
- address wallet：交易用户的钱包合约地址
- bytes32 uuid：委托单唯一标识
- bytes signature：用户对交易的签名授权

撮合流程

- 订单的撮合是由链下系统决定的，链下系统会依照以下顺序撮合交易：

  1. 价格优先（最高买价优先匹配最低卖价，即对吃单方(taker)最有利的价格优先成交），
  2. 时间优先（价格相同且方向相同时按委托时间优先）

- 处理吃单方订单

- 查询每一个挂单方(maker)，处理订单匹配：
    - 比较吃单方与挂单方是否匹配
    - 成功后处理资金转移与股份更新

委托单验证细节：

1. **无签名订单：**
    - 如果没有绑定 EOA，则不做验证，直接交易。安全性低。
    - 有绑定 EOA，则会检查预签名额度是否足够。如果不够，交易回滚，确保用户资产安全。
2. **有签名订单：**
    - 市价单：实际成交金额 ≤ 期望成交金额
    - 限价单：实际成交股份 ≤ 期望股份，且成交价格对用户有利（买单需成交价 ≤ 限价；卖单需成交价 ≥ 限价）

买入和卖出股份都会需要签名验证，来确认这是您本人的行为。签名验证用到的 encodedData 包含以下字段：

- 合约标识（bytes32）
- optionId
- mode
- buysell
- expectedPrice
- expectedShares
- fee
- uuid
- chainId

- 预测合约地址

这些签名资讯确保平台送出的订单必须与您的下单资讯完全一致，也防止了重放攻击和跨链重放。

在向您的钱包合约扣款时，钱包合约还会确认该预测合约使用的预言机是否被您认可。详见 upgradeWallet 章节了解如何升级预言机。

与其他参数一样，手续费同样是由链下传入，包含在签名内容中。因此只要手续费与您的签名有任何不同，整笔交易就会回滚，平台一分钱都没有办法多收。

吃单方与挂单方匹配规则

1. 吃单方买：是（buy yes）

- 可匹配：挂单方买否（buy no）或卖是（sell yes）
- 匹配条件：
    - 买是 vs 买否：价格相加等于 1 USDC
    - 买是 vs 卖是：价格必须相等

2. 吃单方为买：否（buy no）

- 可匹配：挂单方买是（buy yes）或卖否（sell no）
- 匹配条件：
    - 买否 vs 买是：价格相加等于 1 USDC
    - 买否 vs 卖否：价格必须相等

3. 吃单方为卖：是（sell yes）

- 可匹配：挂单方买是（buy yes）或卖否（sell no）
- 匹配条件：
    - 卖是 vs 买是：价格必须相等
    - 卖是 vs 卖否：价格相加等于 1 USDC

（请注意：卖是与卖否成交，代表股份清 0）

4. 吃单方为卖：否（sell no）

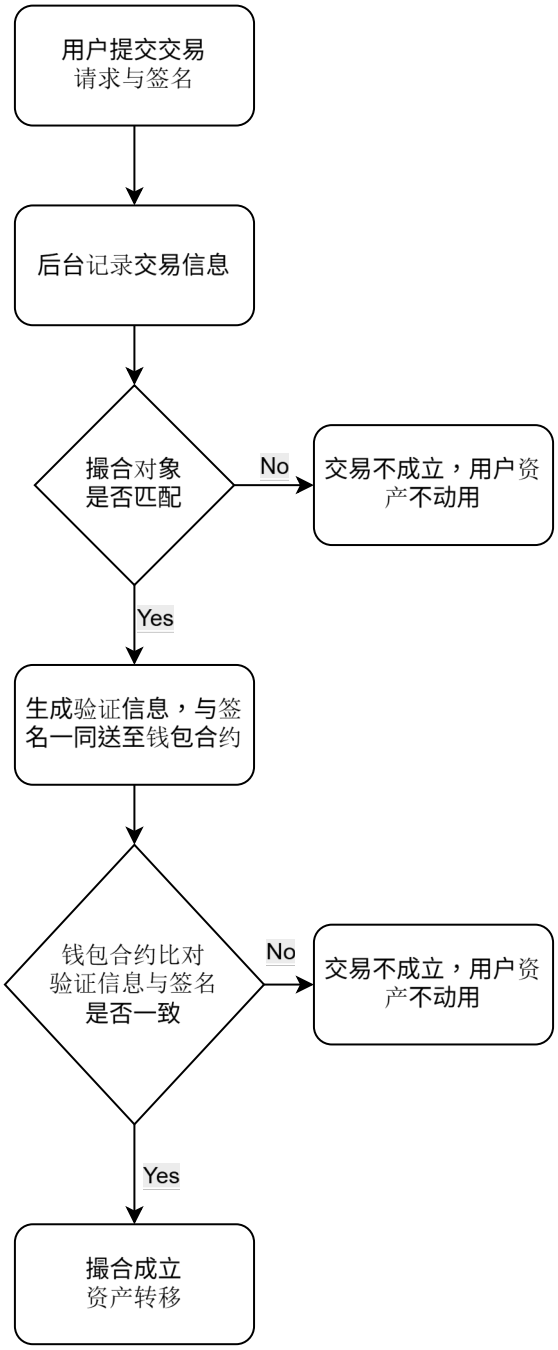
- 可匹配：挂单方买否（buy no）或卖是（sell yes）
- 匹配条件：
    - 卖否 vs 买否：价格必须相等
    - 卖否 vs 卖是：价格相加等于 1 USDC

（请注意：卖否与卖是成交，代表股份清 0）

总结：

- 搓合失败：无法扣款。
- 搓合成功：签名验证，交易对用户有利。
- 转移金额：符合签名授权范围。
- 转移后安全：股份与资产同步结算，确保领取奖励无风险。

流程图如下：



结算(settle)

每当预言机公布结果后，平台会调用 settle 方法，对用户持有的股份进行结算。结算后，会将用户应得的金额，直接从预测合约转到用户的钱包合约。

参考代码：

function name: settle

input:
1. uint256 requestId，服务器用于识别本次交易 ID。
2. uint256 optionId，识别要结算的预测选项。
3. address[] wallets，要发放盈利的用户钱包地址数组。

output:
无返回值

结算逻辑

在调用 settle 方法时，预测合约通过调用预言机合约的 getOutcome 方法获取结果。从预言机获取的结果有以下可能：

- **结果 = 0（结果尚未出炉）**

  结果尚未出炉，交易回滚(revert)。

- **结果 = 1（是，Yes）**

  用户每持有 1 股「是」的股份，可以领取 1 USDC。

- **结果 = 2（否，No）**

  用户每持有 1 股「否」的股份，可以领取 1 USDC。

- **结果 = 3（平局）**

  无论是或否，每股都可以领取 0.5 USDC。

  （注意：无论当时价格是多少，只要是 3 或 5，每股固定发放 0.5 USDC）

- **结果 = 4（待仲裁）**

  交易回滚(revert)。

- **结果 = 5（交易市场取消）**

  无论是或否，每股都可以领取 0.5 USDC。

  （注意：无论当时价格是多少，只要是 3 或 5，每股固定发放 0.5 USDC）

用户自行结算(settle)

为了防止极端情况下（如平台终止服务）用户资产无法取回，PickYesNo 允许用户自行调用结算功能，取回自己的资产。

此方法与官方结算规则完全相同，但需要用户自行承担 gas 费用。因此，即使平台终止服务，用户也可以通过这个方法拿回应得的利润。

参考代码：

function name: settle

input:
1. uint256 optionId，要结算的预测选项。
2. address wallet，发放盈利的目标地址（必须持有该 optionId 的股份）。

output:
无返回值

平台可挪用的金额

从上面的介绍中我们能看到，结算后，会将用户应得的金额从预测合约内，转移到用户的钱包合约。而这些金额是在搓合(broker)时，从用户的钱包合约中转进预测合约的。

那项目方有没有可能在结算前，就把预测合约内的 USDC 领走，导致结算后奖励发不出来？PickYesNo 的合约代码完全杜绝这种事发生的可能。

函数 transferToFee 是除了结算之外，唯一能将 USDC 从预测合约转出的途径。从 transferToFee 的代码中，您可以明确看到我们转出的 USDC 无法超过 fees 这个门槛。而 fees 是交易搓合时明确留给项目方的手续费。

所以，平台确实能从预测合约转出 USDC，但仅限于搓合时明定的手续费。

参考代码:

function name: transferToFee
input:
1. uint256 requestId，服务器用于识别本次交易 ID。
2. uint256 amount，欲从预测合约转出的 USDC 数量。
output:
无返回值

取得预测市场资讯(getSetting)

预测市场中每个选项的资讯都会存储在区块链上，您可以调用 getSetting 方法来查询。
这是个 view 方法，调用它不需要消耗 gas 费。其实它回传的就是 PredictionSetting 这个自定义 Struct 的内容。

针对不同形式的预测市场，这个函数的输入输出有不同意义，下面我们分两种情境来解释。

参考代码:

function name: getSetting

加密币预测市场

input:
1. uint256 optionId，在加密币预测市场，optionId 代表的是轮次。

output:
回传我们自定义的资料结构 PredictionSetting。在加密币预测市场时，每项的意义如下：
1. uint64 optionId，预测轮次。每期加密币预测市场，仅有结束时间不同。为了避免部署重复的合约，我们用轮次代表不同结束时间的预测市场。

2. uint64 roundNo, 在加密币预测市场无意义。
3. uint32 startTime, 加密币预测市场结束时间的基准值。实际结束时间点为: startTime + interval * optionId。
结束后，用户就不能再买卖股份。
4. uint32 endTime，在加密币预测市场无意义。如果这个值为0，就是加密币预测市场。
5. uint32 interval, 每个轮次的间隔时间。
6. uint16 aggregator, 不同数字代表不同的加密货币币别。在加密币市场不为0。
7. uint16 maxVotes, 在加密币预测市场无意义。
8. uint64 stakingAmount，在加密币预测市场无意义。
9. uint64 challengeStaking，在加密币预测市场无意义。
10. uint32 votingDuration，在加密币预测市场无意义。
11. uint32 challengeDuration，在加密币预测市场无意义。
12. uint32 totalRewards，在加密币预测市场无意义。
13. uint8 rewardRanking，在加密币预测市场无意义。
14. uint8 challengePercent，在加密币预测市场无意义。
15. bool independent，在加密币预测市场无意义。

一般预测市场

input:
1. uint256 optionId，要查看资讯的预测选项。

output:
回传我们自定义的资料结构 PredictionSetting。在一般预测市场时，每项的意义如下：
1. uint64 optionId，预测选项。
2. uint64 roundNo, 预测轮次。以彩票来说，每期彩票都有完全一样的胜利条件和号码。为了避免不断部署重复的合约，我们在同一份合约中加入"轮"的概念，以代表不同期的彩票时间点。
并非所有预测合约都有多个轮。因为每次选举间隔时间过长，且候选人都不同，这种情况下，一份预测合约就只会有一轮，也就是只对应到一个预测市场。
3. uint32 startTime, 在一般预测市场无意义。如果这个值为0，就是一般预测市场。
4. uint32 endTime，预测市场结束时间点的unix timestamp。
5. uint32 interval, 在一般预测市场无意义。如果这个值为0，就是一般预测市场。

6. uint16 aggregator, 在一般预测市场无意义。如果这个值为0，就是一般预测市场。
7. uint16 maxVotes, 每个选项最多能投几票。
8. uint64 stakingAmount，投票质押金额。因为USDC的decimal是6，所以您拿到的数字要除以**6才是实际的USDC量。例如当您看到合约回传10**8，那是代表100USDC。
9. uint64 challengeStaking，挑战质押金额。因为USDC的decimal是6，所以您拿到的数字要除以**6才是实际的USDC量。例如当您看到合约回传10**8，那是代表100USDC。
10. uint32 votingDuration，投票时长（单位秒）。
11. uint32 challengeDuration，挑战时长（单位秒）。
12. uint32 totalRewards，总奖金。因为USDC的decimal是6，所以您拿到的数字要除以10**6才是实际的USDC量。例如当您看到合约回传10**8，那是代表100USDC。
13. uint8 rewardRanking， 排名多少才有奖励。
14. uint8 challengePercent， 挑战者分走奖金的百分比：0~100%。
15. bool independent， 是否为独立结果，true=独立结果，false=单一结果。

修改预测市场资讯(setSetting)

当要添加新选项，或是发现原本的选项参数设定错误时，平台会调用setSetting函数修改设定。

平台不能任意修改设定，而是有以下限制：

1. 新增选项的optionId > 0

2. 倘若选项之前不存在，添加新的optionId，并依照输入设定属性roundNo、endTime、votingDuration和challengeDuration。其他属性则套用预设值。

3. 倘若选项之前已存在，只有在预言机的getOutcome回传0，也就是尚未有结果时才能修改，且仅能修改endTime、votingDuration和challengeDuration三个属性。

参考代码:

function name: setSetting

input:
我们自定义的资料结构 PredictionSetting 阵列，其中每一项的意义能参考上面的 getSetting 函数说明。
output:

无

Chainlink 预言机合约

Chainlink 预言机合约是专门为加密币预测设计的，我们用函数 savePrice，直接将 Chainlink 公平、公开、无法窜改的价格数据存储到链上，函数 getOutcome 会以此决定最终结果。

存储价格(savePrice)

函数 savePrice 能从 Chainlink 获取特定轮次的币价，将之存到链上。

这里我们需要定义两个名词:

1. 轮次

在加密币预测市场中，所有预测题目的形式皆为:
某一加密货币在第 n 个轮次的价格，相较于第 n-1 个轮次，是上涨还是下跌。

每期加密币预测市场，仅有结束时间不同。为了避免部署重复的合约，我们用轮次代表不同结束时间的预测市场。与 Chainlink AggregatorV3Interface 中定义的 roundId 完全不同。

2. 第 n 个轮次的价格

要知道第 n 个轮次的价格，我们要先知道结束时间。结束时间为:

startTime + interval * optionId (详见测合约 getSetting startTime)，

这个结果是 unix timestamp。我们会透过迭代的方式，找到 Chainlink 所有已写入的数据中，**在这个时间点之后，最接近这个时间点(大于等于)**的价格，当作第 n 个轮次的价格。

如果没有找到，可稍后用不同的输入再次寻找，对结果不会有任何影响。

我们的合约能判断 Chainlink 是否已写入那个时间点的价格，只有已写入的价格会被储存。

参考代码:

function name: savePrice

input:
1. uint256 requestId，服务器用于识别交易 ID。
2. address prediction，预测合约地址。
3. uint256 optionId, 要存储第几个轮次的价格。
4. uint80 startRoundId, 当这个参数为 0，我们会从 Chainlink 最新的数据开始找目标价格；如果不为 0，我们会把这个参数当作起始 RoundId 开始找目标价格。
5. uint256 length, 最大迭代次数。如果超过这个次数仍未找到目标价格，交易回滚。

output:
无

获取结果(getOutcome)

要用 getOutcome 获取第 n 个轮次的结果，需要先调用 savePrice，保存第 n、n-1 个轮次的价格。

参考代码:

function name: getOutcome

input:
1. address prediction，预测合约地址。
2. uint256 optionId, 轮次。

output:

1. uint256 outcome, 有以下可能：

    0:仍在等待 savePrice 函数保存轮次 optionId，和 optionId-1 轮的价格。

    1: 轮次 optionId 的价格 >= optionId-1 的价格。

    2: 轮次 optionId 的价格 < optionId-1 的价格。

    5: 如果结束时间超过 7 天都没有记录到需要的价格，则判定取消，以便让预测市场中持有股份的用户能进行结算。

乐观预言机合约

乐观预言机合约至关重要，除了加密币市场外，所有预测市场的结果都是由它决定。

与加密币预测市场不同，一般预测市场没有类似 Chainlink 的链上数据来源。为了确保预言机仍能给出公平、公正、公开的答案，PickYesNo 精心设计了乐观预言机，确保每一个结果都与事实相符！

PickYesNo 的乐观预言机采用去中心化的架构，用户通过投票，选出他们认为正确的市场结果。而我们创新的挑战机制，让用户对投票结果感到质疑时，可以进行挑战，让平台仲裁。平台会奖励在投票和挑战阶段选出正确结果的用户，感谢他们为平台的公平公正做出贡献。我们相信这样的作法，可以让预言机给出最公正的结果，避免在金融市场中，被资金实力强大的个人或组织用资本颠覆事实。

预测市场类型

预言机合约针对不同预测合约类型，有不同的处理方式，以下我们先介绍单一结果和独立结果：

- **单一结果**

  在同一轮中，不同选项互斥，只能有一个结果发生。

  举例：2049 年 X 国总统大选，A、B、C 三位候选人中，只能有一人胜选，不会出现多人同时胜选。

- **独立结果**

  在同一轮中，不同选项可以同时发生，互不影响。

  举例：2049 年 X 国大选，Y 党的候选人在 A、B、C 州分别胜选与否，可能零胜、单胜、多胜或全胜。

投票

在预言机合约的投票轮期间，用户通过预言机投票，决定预测市场的结果。投票轮的定义会在后面详细介绍。

不同于预测合约只有“是”与“否”，预言机合约的结果还有一个额外的可能：不明确 50-50 结算。不明确 50-50 结算的意义是：预测市场偶尔会出现一些事前无法预期的状况，不管选是或否都不洽当，这时就能用不明确 50-50 结算当作最终结果。

独立结果的预测市场，用户在预言机投票时，针对每个选项有三种结果可以选择：是、否、不明确 50-50 结算。在同一轮中，每个选项有各自的运行周期，独立统计各自的票数。一个选项以是、否、不明确 50-50 结算这三种结果中，票数最高的作为最终结果。

单一结果的预测市场，用户在预言机投票时，针对每个选项只有两种结果能选择：是、不明确 50-50 结算。在同一轮中，最多只会有一个选项的结果为是。
在单一结果中，不明确 50-50 结算的票数是共用的，也就是说不管您投给哪个选项的不明确 50-50 结算，这些票数都会累加起来，跟其他选项的"是"的票数做比较。
如果不明确 50-50 结算是最高票，不管您查询同一轮中的哪个选项，结果都是不明确 50-50 结算。如果有一个选项的"是"为最高票，则您查询该选项的结果时，会返还是，其他同一轮中的选项都会返还“否”。
如果最终结果为不明确 50-50 结算，您不管投给哪个选项的不明确 50-50 结算，都视为您投给了正确结果。如果最终结果为某个选项的“是”，只有投给那个选项 “是”的用户，被视为投给了正确结果。

以上说明都是在不需要仲裁的情况下才会成立。关于仲裁的结果，以及何时需要仲裁，将在后面章节详细说明。

为了保证投票的公正性，用户在投票时，要质押一定数量的 USDC。

如果最终确认用户投票的结果与事实相符，质押金将全额返还并依投票顺序额外获得奖励；反之，质押金将被没收。

参考代码：

function name: vote

input:
1. uint256 requestId，服务器用于识别交易 ID。
2. address prediction，投票目标的预测合约地址。
3. uint256 optionId，预测合约中的选项 ID。
4. uint256 outcome，投票结果：1=是，2=否，3=平局。
5. address wallet，用户的钱包合约地址。
6. bytes signature，本次交易的委托签名，用于验证身份。签名内容包含 prediction、optionId、outcome 和质押金额。平台送出的交易只要有一处与签名内容不同，整笔交易就会回滚，确保平台无法伪造您的投票内容，也不能多收一分质押金额。

output:
无返回值

投票限制条件

- 必须处于投票轮期间，后面会详细介绍投票轮的定义。
- 独立结果：每人每个选项只能投票一次。
- 单一结果：每人每个预测市场只能投票一次。
- 钱包合约中必须有足够的 USDC 质押到预言机合约。

若钱包合约有绑定 EOA，平台无法擅自动用投票用户的资产，因为必须有用户签名授权或预签名额度。

获取预测结果

预言机合约可以透过函数 getOutcome 查询当前选项的预测结果。

预言机回传的值是介于[0, 5]的整数，其中 1、2、3 和 5，都是代表最终结果，一旦产生就永恒不变，预言机和预测合约能根据此结果返奖结算。返回 0 或 4 时，会随着时间进程变为最终结果。同样的回传结果会对预测市场和预言机用户带来不同的影响：

	对预测市场用户	对预言机用户
0：尚未有结果(Pending)	无法结算，直到最终结果产生。	无法返奖，直到最终结果产生。
1：是(Yes)	用户每持有一股份 Yes，能获得 1USDC。	仅有在投票和挑战中选择 Yes 的用户能获得奖励。
2：否(No)	用户每持有一股份 No，能获得 1USDC。	仅有在投票和挑战中选择 No 的用户能获得奖励。
3：不明确 50-50 结算(Tie)	用户每持有一股份 Yes，能获得 0.5USDC；每持有一股份 No，能获得 0.5USDC。	仅有在投票和挑战中选择不明确 50-50 结算的用户能获得奖励。

4：待仲裁(Pending Arbitration)	无法结算，直到最终结果产生。	无法返奖，直到最终结果产生。当有验证者发起挑战，或是最高票数相同时，进入此阶段，等待平台仲裁。
5：已取消(Cancelled)	用户每持有一股份 Yes，能获得 0.5USDC；每持有一股份 No，能获得 0.5USDC。	不管投票和挑战选什么选项，都不会获得奖励，但质押金会全额退还。

参考代码：

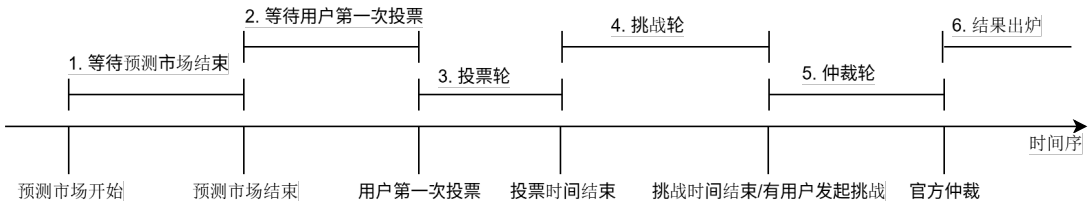
function name: getOutcome

input:
1. address prediction，预测合约地址
2. uint256 optionId，预测合约内的选项 ID

output:
1. uint256 返回的预测结果，可能值是介于[0, 5]的整数，如上述说明。

预言机合约决策流程

针对每一份预测合约，预言机合约的时程可划分为：



各阶段说明

针对不同预测合约不同选项，预言机会有不同流程。

1. 等待预测市场结束

- 在这个阶段,对应的预测市场还没结束,用户与平台皆不得在预言机中执行任何动作。
- 如果预测市场的类型是单一结果，所有选项共用同样的结束时间，如为独立结果，每个选项有各自的结束时间。
- 预测市场结束后，进入阶段 2。

2. 等待用户第一次投票

- 对应的预测市场结束后，预言机会自动进入此阶段，直到第一位用户投票。
- 此阶段最多持续 120 天。在 120 天内，只要没有用户投票，就会一直持续，而后面所有的阶段都会顺延。
- 如果超过 120 天都没有用户投票，跳到阶段 6，并将回传结果设为 5：已取消。这样做的目的是：即使无人参与预言机投票，持有预测合约股份的用户，仍能退回 USDC。
- 如果预测市场的类型是单一结果，只要有一个选项有用户投票，所有选项都会共同离开此阶段；如为独立结果，每个选项都要各自有用户投票，才会离开此阶段。

3. **投票轮**

   - 在第一张票投下后，马上进入投票轮。投票轮的时长是固定的，从第一张票投下时开始计时，固定时间后结束。
   - 参数 votingDuration 决定了投票轮会持续多久，可以在对应的预测合约中用函数 getSetting 查询到。
   - 如果预测市场的类型是单一结果，所有选项共用 votingDuration；如为独立结果，每个选项都有各自的 votingDuration。

4. **挑战轮**

   - 投票时间结束后自动进入挑战轮。有两种可能会导致挑战轮结束:
     1. 在挑战轮期间有用户发起挑战。
     2. 经过 challengeDuration 后，挑战轮自动结束。challengeDuration 一样是预测合约的参数，能在对应的预测合约中用函数 getSetting 查询到。

5. **仲裁轮**

   - 并非所有预测市场都会进入仲裁轮。有以下两种情况发生其一时会进入仲裁轮：
     1. 在挑战轮期间有用户发起挑战，挑战轮立即结束，进入仲裁轮。
     2. 经过 challengeDuration 后，有至少两个选项的票数是相同的，这时会需要平台仲裁，来决定最终结果。
   - 反之，挑战轮期间没有用户发起挑战，且挑战轮结束后没有相同票数，那就不会有仲裁轮，结果将直接出炉。

- 如果进入仲裁轮 120 天后，平台仍没有主动仲裁，进入阶段 6，并将回传结果设为 5：已取消。这样做的目的是：在万一平台终止服务，无法仲裁时，参与预言机和预测合约的用户仍能拿回自己的 USDC。

6. **结果出炉**
   - 最终结果出炉，预测市场能用此结果结算。
   - 倘若没有进入仲裁轮，最终结果就是所有用户投票的最高票。
   - 如有进入仲裁轮，最终结果就是平台的裁定，或是超过 120 天自动回传 5。

判定返回结果的逻辑

在不同阶段调用 getOutcome，会返回不同结果。

1. **返回 0**

   代表尚未有结果。在等待预测市场结束、等待用户第一次投票、投票轮和挑战轮这四个阶段调用 getOutcome，皆会返还 0。

2. **返回 4**

   代表目前处于仲裁轮。

3. **返回 1、2 、3 或 5**

   代表这是预言机给出的最终结果，预测市场能根据此结果进行结算。已经在预言机投票或挑战的用户，也能根据此结果返奖。

挑战

在预测合约的挑战轮期间，用户可以对投票结果提出质疑。

不同于预测合约只有“是”与“否”，预言机合约的结果还有一个额外的可能：不明确 50-50 结算。不明确 50-50 结算的意义是，预测市场偶尔会出现一些事前无法预期的状况，不管选是或否都不洽当，这时就能用不明确 50-50 结算当作最终结果。

独立结果的预测市场，用户在预言机挑战时，针对每个选项有三个结果能选择：是、否、不明确 50-50 结算。

单一结果的预测市场，用户在预言机挑战时，针对每个选项只有两个结果能选择：是、否、不明确 50-50 结算。
如果最终结果为不明确 50-50 结算，您不管挑战选哪个选项的不明确 50-50 结算，都视为您在挑战中给出了正确结果；如果最终结果为某个选项的“是”，只有挑战选那个选项“是”的用户，被视为在挑战中给出了正确结果。

为了防止恶意挑战，用户在挑战时，同样需要质押一定数量的 USDC。
如果最终确认用户挑战的结果与事实相符，质押金将全额返还并获得额外奖励；反之，质押金将被没收。

参考代码：

function name: challenge

input:
1. uint256 requestId，服务器用于识别交易 ID。
2. address prediction，挑战目标的预测合约地址。
3. uint256 optionId，预测合约中的选项 ID。
4. uint256 outcome，用户认为正确的结果：1=是，2=否，3=平局。
5. address wallet，用户的钱包合约地址。

6. bytes signature，本次交易的委托签名。签名内容包含 prediction、optionId、outcome 和质押金额。平台送出的交易只要有一处与签名内容不同，整笔交易就会回滚，确保平台无法伪造您的挑战内容，也不能多收一分质押金额。

output:
无返回值

挑战限制条件

- 必须处于挑战轮期间。
- 挑战次数有限制。
    - 单一结果：所有选项合计仅能挑战一次。
    - 独立结果：每个选项各能挑战一次。
- 挑战时不能选择当前票数最高的选项，如发生平局则不限制。
- 钱包合约中必须有足够的 USDC 质押到预言机合约。
- 平台无法擅自动用投票用户的资产，因为必须有用户签名授权或预签名额度。

仲裁

在以下两种情况，平台可以进行仲裁：

条件	**说明**
用户发起了挑战	单一结果：平台仲裁所有选项 独立结果：平台仅仲裁被挑战的选项
投票出现票数相同	单一结果：有两个或以上的选项同时是最高票

	时，平台仲裁所有选项 独立结果：同一选项下，是/否/平局中如果发生最高票数相同，平台仲裁该选项

仲裁虽然由平台的执行者 EOA 发起，但必须搭配两个专门的仲裁者 EOA 签名才能生效。即使黑客取得执行者 EOA 的私钥，也无法随意进行仲裁。

仲裁者 EOA 只提供签名，不连接网络，最大程度降低私钥泄漏的风险，拥有比执行者 EOA 更高的安全性。每次仲裁更需要两个仲裁 EOA 提供签名，进一步提升了安全性。

参考代码：

function name: arbitrate

input:
1. uint256 requestId，服务器用于识别交易 ID。
2. address prediction，要仲裁的预测合约地址。
3. uint256 optionId，要仲裁的选项 ID。
4. uint256 outcome，平台裁定的结果：1=是，2=否，3=平局，5=取消。

output:
无返回值

奖励

任何人都能调用发放奖金的函数，但在大部分情况，平台都会主动调用，以替用户节省手续费，同时所有的奖金均由平台提供。

即使平台终止服务，用户仍能主动调用，拿回质押的金额。

参考代码：

function name: reward

input:
1. uint256 requestId，服务器用于识别本次交易的 ID。
2. address prediction，要发放奖励的预测合约地址。
3. uint256 optionId，针对的选项 ID。
4. address[] wallets，要发放奖励的用户钱包合约地址数组。

output:
无返回值

奖励发放流程

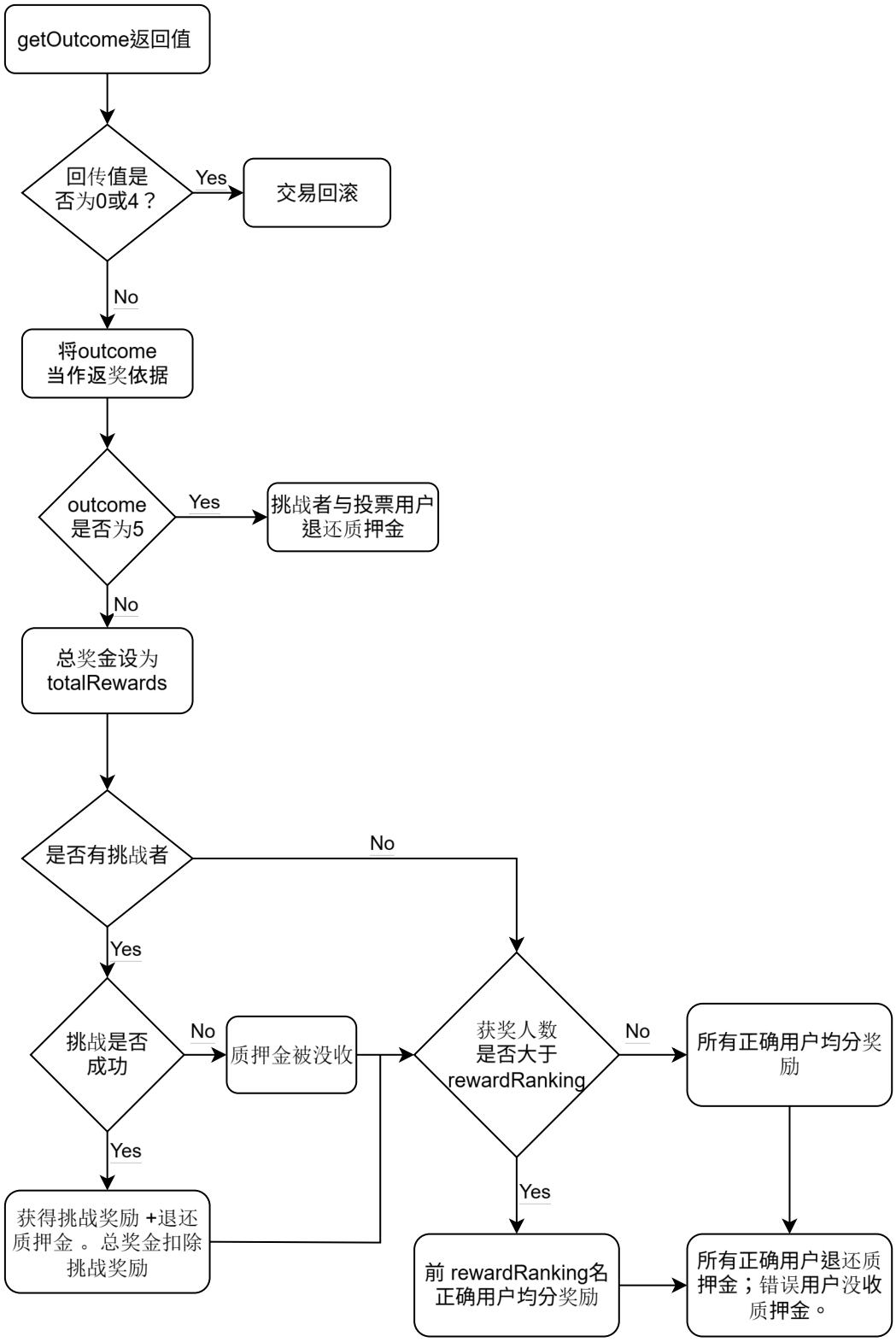
1. 获取结果参数 outcome。函数 reward 会在内部调用_getOutcome，来拿到要用来当作返奖依据的参数 outcome。
    - 如果 outcome 是 0 或 4，交易回滚。
    - 否则判断是否返奖。
2. 判断是否发放奖励。当 outcome 等于 1、2 或 3 时，才会发放奖金。如果 outcome 等于 5，只会退回质押金。
3. 有挑战纪录
    - 挑战成功
        - 根据预测合约设定的奖励比率 challengePercent，计算挑战者可获得的奖励 challengeRewards
        - 挑战成功者可获得：
            - challengeRewards
            - 返还质押的 USDC

- 挑战失败
  - 平台没收挑战者质押的USDC，不发放奖励。
  -
- outcome=5
  - 挑战者退回质押金，不发奖金。

4. 投票用户奖励
   - 如果用户投票结果正确，一律退还质押金，此外：
     - 如果正确人数大于rewardRanking人，则前rewardRanking名投票的用户平均分配奖励。
     - 如果人数少于或等于rewardRanking，所有选择正确结果的用户平均分配奖励。
   - 投票错误用户
     - 用户质押的USDC将被平台没收。

   - outcome=5
     - 所有投票用户退回质押金，不发奖金。
5. 奖金和退还的质押金，会直接从预言机合约转到用户的钱包合约。

奖励发放逻辑简图


getOutcome返回值
回传值是否为0或4？
Yes
交易回滚
No
将outcome当作返奖依据
outcome是否为5
Yes
挑战者与投票用户退还质押金
No
总奖金设为totalRewards
是否有挑战者
No
Yes
挑战是否成功
No
质押金被没收
获奖人数是否大于rewardRanking
No
所有正确用户均分奖励
Yes
获得挑战奖励 +退还质押金 。总奖金扣除挑战奖励
Yes
前 rewardRanking名正确用户均分奖励
所有正确用户退还质押金；错误用户没收质押金。

平台可挪用的金额

从上面的介绍中我们能看到，返奖后，会将用户应得的金额从预言机合约内，转到用户的钱包合约。而预言机内的金额由两部分组成：平台提供的奖金，和用户的质押金(包括投票和挑战)。

那项目方有没有可能在返奖前，就把预言机合约内的 USDC 转走，导致用户领不到应得的奖励呢？预言机合约的代码完全杜绝了发生这种事的可能性。

我们使用变数 stakingAmount 来控制平台能从预言机合约转出的金额，这项变数的设计确保了平台唯一能从合约转出 USDC 的函数 transferToFee，没有办法将用户的质押金转走。

所以，平台确实能从预言机合约转出 USDC，但仅限于平台提供的奖金。因此，万一平台终止服务了，虽然奖金可能不会发放，但用户还是能将投票和挑战时的质押金完全领回来。

参考代码：

function name: transferToFee

input:
1. uint256 requestId，服务器用于识别本次交易 ID。
2. uint256 amount，欲从预言机合约转出的 USDC 数量。

output:
无返回值

权限合约

在 PickYesNo 的智能合约体系中，大多数功能仅限特定的 EOA（外部账户）执行。这些权限身份由权限合约进行集中管理与定义：

1. 手续费地址（feeEOA）

所有智能合约共用一个手续费地址。合约中产生的所有手续费等收益，只能转入该地址。

2. 管理员地址（managerEOA）

所有智能合约共用一个管理员地址。此地址可用于合约的管理操作。

3. 营运者地址(operationEOA)

所有智能合约共用一个营运者地址。此地址用于营运相关事务的操作。

4. 仲裁者地址(arbitratorEOA)

专门负责预言机仲裁的地址。

5. 执行者地址（executorEOA）

每份智能合约允许多个执行者地址并存。用户与合约的绝大多数交互操作，都是通过执行者地址来发起交易。为保障用户资金安全，执行者调用合约前必须获得用户的签名，或用户事先完成预签名额度，否则交易将会自动回滚(revert)。

参考

合约地址:
https://pickyesno.com/docs/contract/contract-address

合约源代码:
https://pickyesno.com/docs/contract/contract-sourcecode

合约审计报告:
https://pickyesno.com/docs/contract/contract-audit-report