

Meson跨链桥合约源码分析

- [概述](#)
 - [核心原理-HTLC](#)
 - [Meson的优点](#)
 - [Meson中的三种角色](#)
 - [swap请求的各种状态](#)
 - [签名](#)
- [合约源码解析](#)
 - [swap的调用流程](#)
 - [pool](#)
 - [Pool owner vs Authorized address](#)
 - [相关合约方法](#)
 - [tokenIndex](#)
 - [跨链swap的接收者](#)
 - [encodedSwap](#)
 - [Swap Id](#)
 - [重要状态变量](#)
 - [重要方法](#)
- [如何查看Meson Explorer](#)

概述

- 官网: <https://meson.fi/>
- github: <https://github.com/MesonFi>
- Meson Explorer: <https://explorer.meson.fi/>
- 文档: <https://docs.meson.fi/>
- 平台收入: 跨链swap时, Meson会在【目标链】收取service fee, 流动性提供者也会在【目标链】收取lp fee。
- Meson目前主要支持的是同一类型币种的互换 (例如稳定币USDC-> BUSD, 主网币锚定币ETH->WETH), 因此兑换时不会有滑点, 只有手续费: `swapOut amout= swapIn amount -(service fee+lp fee)`

核心原理-HTLC

- 哈希时间锁定（Hash TimeLock Contract，简称 HTLC）是密码学方法，交互双方用户通过为自己的数据赋予**哈希锁和时间锁**进行数据交互，完成相应的解锁步骤即可获取对方用来交换的数据。
 - 哈希锁：通过哈希值上锁，上锁之后只有用产生这个哈希值的原本值进行开锁，
 - 时间锁：时间锁要求在规定时间内输入哈希锁的密码。如果时间锁的时间是 1 个小时，那么就要求用户需要在 1 个小时内输入哈希锁的密码，如果在 1 个小时后输入哈希锁的密码，时间锁仍然不会开启。
- 简单讲，就是在智能合约的基础上，双方先锁定资产，如果都在有限的时间内输入正确哈希值的原值，即可完成交易。如果时间超过了规定时间，锁定在系统中的代币将会被原路收回，即交易失败，整个过程用户不会损失资产。
- 举例说明整个过程：

现在大白将利用哈希时间锁定的机制把自己的比特币在小黑那里兑换以太币，具体的操作步骤如下：

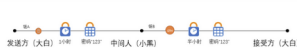
(1) 大白先生成随机数 S，再把随机数的哈希值 H(S) 通过网络给小黑，假设随机数是 123，哈希值是 a03a。

同时，大白进行时间上锁和哈希上锁，假设时间锁的时间为1小时，哈希锁上锁的哈希值是 a03a。上完锁后，待转换的比特币就被锁定在链 A 上。



(2) 小黑收到大白给的哈希值“a03a”后，小黑根据这个哈希值在以太坊上部署智能合约，并往合约中存同等价值的以太币。小黑的智能合约要求大白在规定时间内提供密码“123”才可以取走智能合约中的以太币。

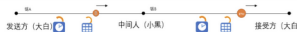
这个过程相当于，小黑自己也上了两把锁，其中哈希锁和大白的那把哈希锁一样，需要用同样的密码才可以打开，时间锁假如为半个小时。



(3) 大白使用小黑的这个智能合约，并在半个小时内输入自己的密码“123”，就能打开小黑在链B上的哈希锁，就能取走小黑智能合约里的以太币（相当于小黑的以太币，因为智能合约是小黑创建的，合约里的以太币也是小黑转进去的）。



(4) 因为大白在调用了小黑的智能合约时输入了密码，因此小黑也就知道了密码是“123”，他只要在一个小时内通过这个密码打开链A上的哈希锁，大白的比特币就会转给小黑。



通过上面的过程，可以看到，大白可以通过哈希时间锁定这种方式，实现了比特币到以太币的兑换。当然这往往需要大白多支付一点比特币给小黑作为手续费，毕竟天下没有免费的劳动力。

- Meson跨链方案中，采用的就是HTLC。与上述例子的区别是，Meson中采用的哈希锁不是某个值的hash、而是swap initiator签署的releaseSignature（下方会介绍）

Meson的优点

1. 速度快：跨链swap通常在1~2min内完成。只需等待起始链、目标链的交易确认、无需等待跨链桥的确认
2. 更低的成本：从协议设计到合约实现都进行了大量优化，支持元交易（用户只需签名、无需支付gas fee-->其实支付的手续费中也会包含gas fee）
3. 资金更安全：在swap过程中，不依赖跨链桥、预言机等第三方服务，资金无需锁在跨链桥的资金池中

- 注：Meson并不是完全安全的，具体可能存在的安全问题可参考：

<https://docs.meson.fi/protocol/security>

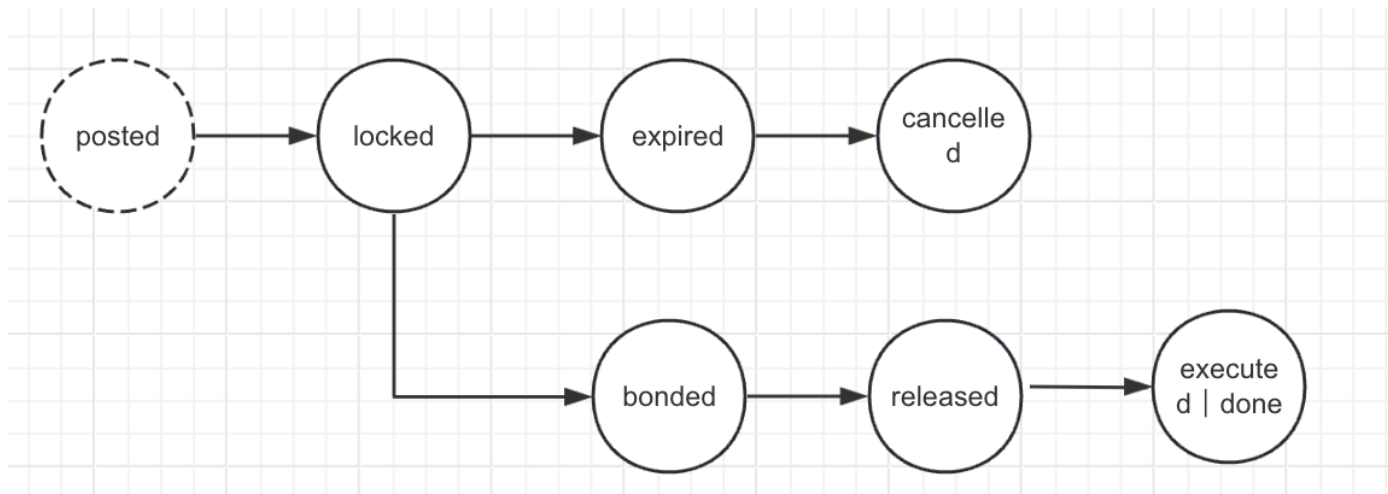
Meson中的三种角色

1. 用户：swap initiator，swap交易的发起者
2. 流动性提供者：Liquidity Provider（LP），为Meson提供流动性，匹配用户的swap请求、执行后续的swap操作

3. 中继器：Relayer，接收用户的swap请求、广播转发给LP

swap请求的各种状态

Meson采用了Atomic Swap技术，这意味着资产不必经过第三方之手，也不会被困在合约中。但一笔交易中会存在多个中间状态



1. posted: swap请求被用户自己提交到链上（如果是relayer代为提交的话，会直接变为bonded状态）
2. bonded: swap请求被绑定到某个pool、由LP (pool owner or authorized addresses)负责执行
3. locked: 对于【起始链】的swap请求来说提交之后、过期或执行之前都属于Locked（表示该资金被锁定在合约里了）；对于【目标链】来说，表示LP已将swap out所需的资金锁定在了合约里
4. expired: swap请求已过期、仍未完成匹配
5. cancelled: swap请求在过期后，可以被取消、从而取回swap in的资产
6. released: 【目标链】的swap out资金释放给用户
7. executed | done: 该swap请求已执行完成（起始链的swap in资金会释放给LP）

签名

- 在整个swap流程中，用户需要提供两种签名：

1. RequestSignature: 用户在【起始链】发起swap请求时，如果由Relayer代为提交交易的话，则需要向Relayer提供一个签名，用于进行签名的摘要信息：

```
digest=keccak256(abi.encodePacked(XXX_SIGN_HEADER or REQUEST_TYPE_HASH, encodedSwap))
```

2. ReleaseSignature: 用户在【目标链】提取swap out的token时，需要向LP提供一个签名证明（注：需要先验证LP已经在目标链上锁定了资金，才能提供签名信息）、这样LP在起始链也可以用同样的签名参数、取出用户swap in的资产。用于进行签名的摘要信息：

```
digest=keccak256(abi.encodePacked(XXX_SIGN_HEADER or RELEASE_TYPE_HASH, encodedSwap, recipient))
```

- 以上两个签名对应的摘要信息可通过API接口 <https://relay.meson.fi/api/v1/swap> 来获取

SDK Integration	
API Integration	
Supported Chains & Tokens	
Tutorials	▼
API Playground	
Advanced	▼
Call a Smart Contract on the Destination Chain	
Initiate a Swap from a Smart Contract	
API	▼
Environments	
GET	List supported chains
GET	Limits for swaps
POST	Get price for a swap
POST	Encode a cross-chain swap
POST	Submit an encoded swap with signatures
POST	Submit swap from smart contract
GET	Check swap status

▼ Schema

▼ result object

encoded hex_string_32_bytes

The encoded swap data which will be used in the [next API](#)

fromAddress address

Same as in the request body if it is an externally owned address (EOA)

fromContract address

Same as **fromAddress** in the request body if it is a contract address

recipient address

Same as in the request body

► **fee** object

▼ dataToSign object[]

Possible values: `>= 2`, `<= 2`

Present if **fromAddress** is an externally owned address (EOA). The data (two **hash**'es) to be signed before submitting a cross-chain swap through the [next API](#).

Array [

message hex_string

hash hex_string_32_bytes

]

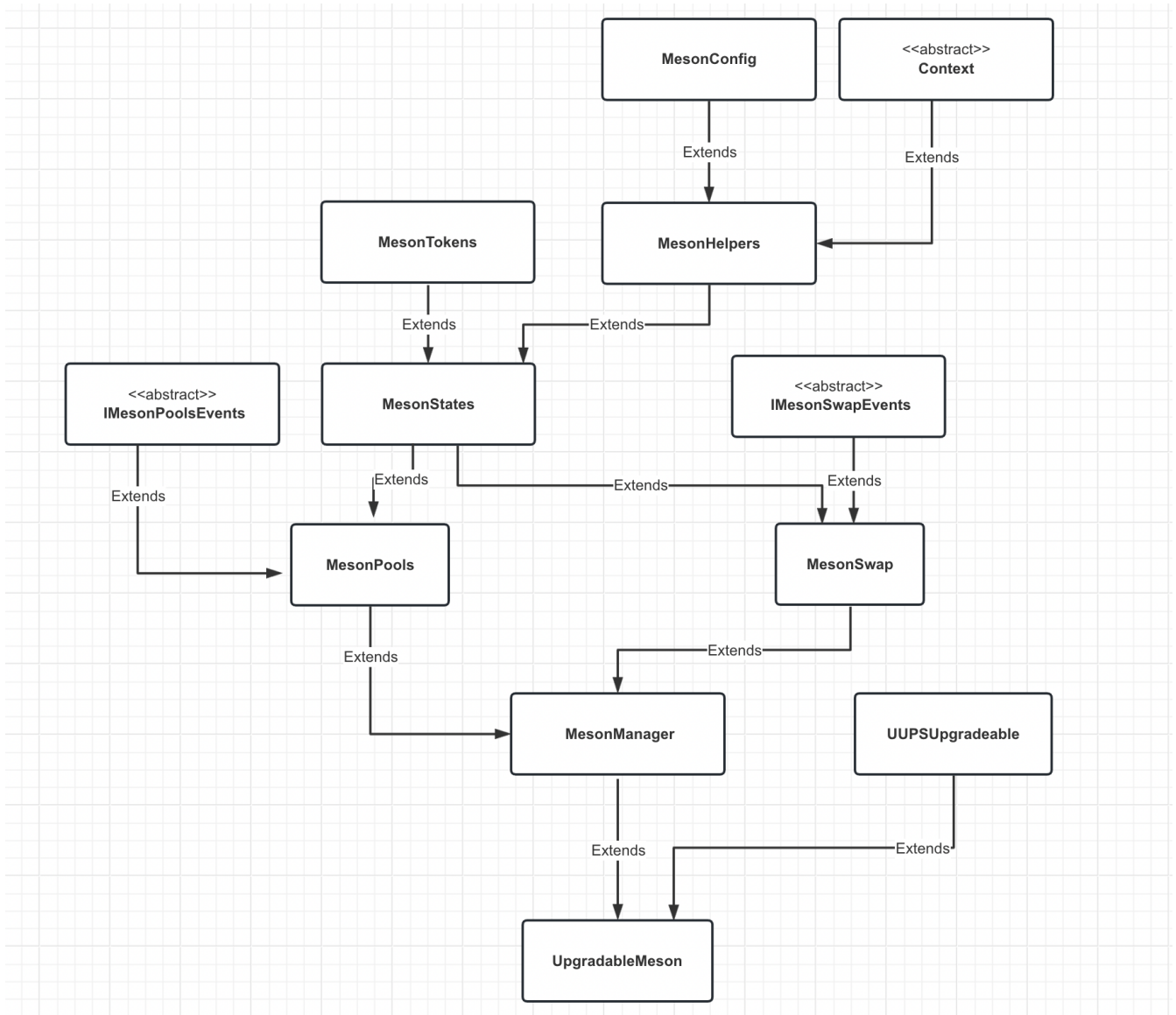
initiator address

Present if **fromAddress** is a contract address. See [usage instructions](#).

合约源码解析

- 合约源码: <https://github.com/MesonFi/meson-contracts-solidity>

- 合约整体架构如下图所示：

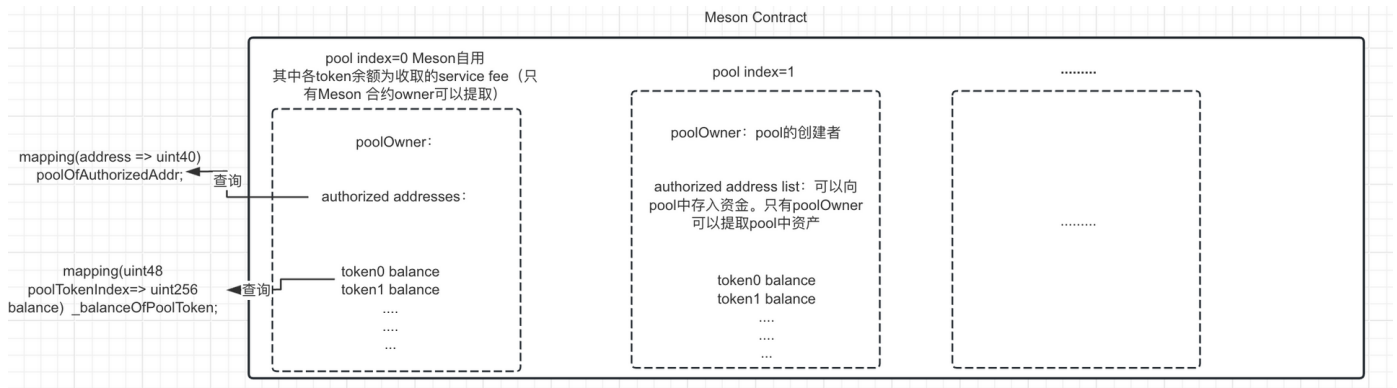


- Meson跨链桥合约采用的是UUPS代理模式
- 代理合约（未开源）：所有EVM链的合约地址都是
0x25ab3efd52e6470681ce037cd546dc60726948d3
- 代理合约指向的Impl合约（已开源，目前所有EVM地址均相同）：
0x7ecc64a774e728aedc65effbe9183c8b5069d149#writeContract

swap的调用流程

1. 【起始链】发布并绑定swap请求： `postSwap()` 提交swap请求--->起始链：【可选，由pool owner 或 authorized addresses调用】 `bondSwap()` 为swap请求绑定对应的pool，即补充完善_postedSwaps中的poolIndex信息
2. 【目标链】LP锁定swap out资金： `lockSwap()` 锁定1.中绑定的pool中的对应swap out资产
3. 【目标链】Meson合约释放swap out资金： `release()`，用户需提供release signature、会收到swap out token
4. 【起始链】LP接收初始swap in资金： `executeSwap()`，LP拿着3.中的release signature获取到1.中用户存入的资金

pool



- Meson中，流动性提供者LP负责匹配用户提交的swap订单，任何注册为Meson LP的地址都会被指定为某个pool的所有者。当用户将指定资金发送、锁定到起始链后，LP需要在目标链上的pool中锁定对应的资金

Pool owner vs Authorized address

- 每个pool都有一个owner，只有owner可以从pool中提取资产。可执行的操作有：
 - depositAndRegister
 - deposit
 - withdraw
- pool owner可以授权其他人作为授权地址Authorized address来代表其执行swap。Authorized address在任何情况下都不允许提取资产、但可以存入资产。可执行的操作有：
 - deposit
 - bondSwap
 - lock
 - release
 - executeSwap
- 通过授予Authorized address角色，LP在存入流动性后、不在需要每次都是用owner私钥来进行swap操作，提高账户安全性
- pool owner默认会被添加为Authorized address。如下代码所示，如果一个LP依次创建了多个pool，只有最后一个pool里、会被默认添加为Authorized address，之前的设置都被覆盖掉了

```
poolOfAuthorizedAddr[poolOwner] = poolIndex; //覆盖之前的值
```

相关合约方法

1. depositAndRegister(uint256 amount, uint48 poolTokenIndex)

- 参数：
 - amount: 存入的token数量

- poolTokenIndex: tokenIndex:uint8|poolIndex:uint40, 其中tokenIndex表示要存入的token index、poolIndex为要创建的pool Index (合约中会校验pool Index的唯一性)
 - 本方法的作用: 注册创建一个新的pool、并在pool中存入资金 (因为pool是虚拟的, 所以token转入的是Meson合约、只是更新了一下pool的余额。token流向: msg.sender-->Meson合约)
2. deposit(uint256 amount, uint48 poolTokenIndex)
- 本方法的作用: 向poolTokenIndex指定的pool中存入token, 数量为amount (token流向: msg.sender-->Meson合约)
 - 注: 并不是所有人都可以调用此方法, 交易发起者必须是AuthorizedAddr, 即
`poolOfAuthorizedAddr[msg.sender]==poolTokenIndex对应的poolIndex`
3. withdraw(uint256 amount, uint48 poolTokenIndex)
- 本方法的作用: 从poolTokenIndex指定的pool中取出token, 数量为amount (token流向: Meson合约-->msg.sender)
 - 注: 只有pool owner可以调用此方法, 用于取出累积在pool中的资金
4. addAuthorizedAddr(address addr)、removeAuthorizedAddr(address addr)
- 本方法的作用: 为poolTokenIndex指定的pool 追加或移除 AuthorizedAddr
 - 注: 只有pool owner可以调用此方法
5. transferPoolOwner(address addr): 转移自己的pool owner权限给addr (上方提到过, 每个地址最多只能作为一个pool的owner, 因此此处无需在参数中指定poolIndex, 可以从合约中直接查询到)
6. lockSwap(uint256 encodedSwap, address initiator)
- 本方法需要在swap的【目标链】上调用, 交易发起者必须是与encodedSwap绑定的那个pool的AuthorizedAddr
 - 作用: 在目标链上从pool中锁定本次swap out所需的token (此处计算token数量时、已扣除lp fee部分。后期会释放给用户)
7. lock(uint256 encodedSwap, bytes32 r, bytes32 yParityAndS, address initiator)
- 作用同lockSwap、该方法已过期、不建议调用
8. unlockSwap(uint256 encodedSwap, address initiator)
- 作用: swap的用户在锁定期间内、未能在【目标链】上提供release signature提取swap out资产的话, 解锁6.中pool锁定的资产
9. release(uint256 encodedSwap, bytes32 r, bytes32 yParityAndS, address initiator, address recipient)
- 参数中, r和yParityAndS为swap initiator签署的release Signature。
 - 作用: 校验用户提供的release Signature, 校验通过后将LP之前锁定的token转给用户 (会先扣除service fee。token流向: Meson合约-->swap recipient, Meson合约-->pool)
 - 所有人都可以调用此方法、帮助完成该swap请求在目标链资金的释放

10. directRelease(uint256 encodedSwap,bytes32 r,bytes32 yParityAndS,address initiator,address recipient)

- 将unlockSwap()和release() 合并到一起执行 (token流向: pool-->swap recipient)
- 所有人都可以调用此方法, 帮助完成该swap请求在目标链资金的释放

11. simpleRelease(uint256 encodedSwap, address recipient)

- 作用同10。主要区别是, 交易发起者必须是合约中设置过的premium manager

tokenIndex

- Meson目前主要支持的跨链币种为主网币、主网币锚定币、稳定币, tokenIndex为Meson为所支持的tokens定义的索引
- 各token对应的索引Index的分配如下所示

```
/// `tokenIndex` in range of 1-255
/// 0:      unsupported
/// 1-32:   stablecoins with decimals 6
/// 1, 9:   USDC, USDC.e
/// 2, 10:  USDT, USDT.e
/// 3:      BUSD, USDT.e
/// 17:     PoD USDC
/// 18:     PoD USDT
/// 19:     PoD BUSD
/// 32:     PoD
/// 33-64:  stablecoins with decimals 18
/// 33:     USDC
/// 34:     USDT
/// 35:     BUSD
/// 36:     (reserved for DAI)
/// 37:     cUSD (Celo)
/// 52:     XDAI
/// 65-128: (Unspecified)
/// 129-190: (Unspecified)
/// 191:    No-swap core
/// 192-247: (Unspecified)
/// 248-251: BNB & BNB equivalent
/// 248:    PoD BNB
/// 250:    (reserved for ERC20 BNB)
/// 251:    BNB as core
/// 252-255: ETH & ETH equivalent
/// 252:    PoD ETH
/// 254:    (reserved for ERC20 ETH like WETH)
/// 255:    ETH as core
```


- tokenDecimals: 如上所示, 1~32的decimals=6, 大于32的部分token的decimals=18, 因此可以根据tokenIndex判断token的decimals

//实际swap时, 参数中token数量均以decimals=6来记录, 在实际执行合约时, 再根据tokenIndex判断token的实际decimals、来决定要不要*1e12

```
function _needAdjustAmount(uint8 tokenIndex) private pure returns (bool) {
    return tokenIndex > 32;
}
```

- tokenType: 如上图所示, 0<tokenIndex<65时, token为稳定币; 65<=tokenIndex<192时, 暂未使用; tokenIndex>=192时非稳定币, 248~251属于BNB及BNB锚定币、因此属于同一类别, 252~255属于ETH及ETH锚定币、因此属于同一类别。在各xxxSwap()中都会通过下图中的modifier verifyEncodeSwap校验swapIn、swapOut的币种为同一类别。

```
function _tokenType(uint8 tokenIndex) internal pure returns (uint8) {
    if (tokenIndex >= 192) {
        // Non stablecoins
        return tokenIndex / 4;
    } else if (tokenIndex < 65) {
        // Stablecoins
        return 0;
    }
    revert("Token index not allowed for swapping");
}
```

```
modifier VerifyEncodedSwap(uint256 encodedSwap) {
    require(_inChainFrom(encodedSwap) == SHORT_COIN_TYPE, "Swap not for this chain");
    require(
        _tokenType(_inTokenIndexFrom(encodedSwap)) == _tokenType(_outTokenIndexFrom(encodedSwap)),
        "In & out token categories do not match"
    );
    require(_postedSwaps[encodedSwap] == 0, "Swap already exists");

    require(_amountFrom(encodedSwap) <= MAX_SWAP_AMOUNT, "For security reason, amount cannot be greater than 100k");

    uint256 delta = _expireTsFrom(encodedSwap) - block.timestamp;
    require(delta > MIN_BOND_TIME_PERIOD, "Expire ts too early");
    require(delta < MAX_BOND_TIME_PERIOD, "Expire ts too late");
} -;
```

- 如上方定义所示, “52:XDAI(以太坊侧链)、251: BNB as core、255: ETH as core”标识的是主网币, 因此可以通过tokenIndex判断token是否为主网币

```
function _isCoreToken(uint8 tokenIndex) internal returns (bool) {
    return (tokenIndex == 52) || ((tokenIndex > 190) && ((tokenIndex % 4) == 3));
}
```

- 平台服务费service fee: 如果tokenIndex>=191, 最小服务费为SERVICE_FEE_MINIMUM_CORE(=5000), 否则最小服务费为SERVICE_FEE_MINIMUM(=500_000)

```

function _serviceFee(uint256 encodedSwap) internal pure returns (uint256) {
    uint256 minFee = _inTokenIndexFrom(encodedSwap) >= 191 ? SERVICE_FEE_MINIMUM_CORE :
    SERVICE_FEE_MINIMUM;
    // 根据swap in数量计算服务费
    uint256 fee = _amountFrom(encodedSwap) * SERVICE_FEE_RATE / 10000;
    // 根据最小服务费修正实际应该要收取的服务费数量
    return fee > minFee ? fee : minFee;
}

```

- tokenIndex和tokenAddress之间的映射关系，保存在以下两个变量中，可以通过tokenForIndex()和indexOfToken()进行查询

```

/// key: the supported token's contract address. address(1)对应主网币
/// value: `tokenIndex` in range of 1-255
mapping(address => uint8) public indexOfToken;

mapping(uint8 => address) public tokenForIndex;

```

跨链swap的接收者

- 如果接收者是个合约，则必须实现IDepositWithBeneficiary接口的depositWithBeneficiary()：需要接收合约主动从Meson合约中transfer token，Meson只会进行approve、不会主动transfer的！

```

/// @notice Transfer tokens to a contract using `depositWithBeneficiary`
/// @param tokenIndex The index of token. See `tokenForIndex` in `MesonTokens.sol`
/// @param contractAddr The smart contract address that will receive transferring tokens
/// @param beneficiary The beneficiary of `depositWithBeneficiary`
/// @param amount The value of the transfer (always in decimal 6)
/// @param data Extra data passed to the contract
function _transferToContract(
    uint8 tokenIndex,
    address contractAddr,
    address beneficiary,
    uint256 amount,
    uint64 data
) internal {
    require(Address.isContract(contractAddr), "The given recipient address is not a contract");
    if (_needAdjustAmount(tokenIndex)) {
        amount *= 1e12;
    }

    if (_isCoreToken(tokenIndex)) {
        // Core tokens (e.g. ETH or BNB)
        IDepositWithBeneficiary(contractAddr).depositWithBeneficiary{value: amount}(
            address(0),
            amount,
            beneficiary,
            data
        );
    } else {
        // Stablecoins
        address token = tokenForIndex[tokenIndex];
        require(Address.isContract(token), "The given token address is not a contract");

        IERC20Minimal(token).approve(contractAddr, amount);
        IDepositWithBeneficiary(contractAddr).depositWithBeneficiary(
            token,
            amount,
            beneficiary,
            data
        );
    }
}

```

encodedSwap

- encodedSwap是uint256类型，将一次跨链swap所需的信息组装在一起
- 其组成：



```
version:uint8|amount:uint40|salt:uint80|fee:uint40|expireTs:uint40|outChain:uint16|outToken:uint8|inChain:uint16|inToken:uint8
```

- uint8 version表示的是编码版本，计算公式： $\text{version} = \text{uint8}(\text{encodedSwap} \gg 248)$;
- uint40 amount表示的是swap in数量，计算公式： $\text{amount} = (\text{encodedSwap} \gg 208) \& 0xFFFFFFFF$
- uint80 salt：其中包含多种信息，如下所示。计算公式： $\text{salt} = \text{uint80}(\text{encodedSwap} \gg 128)$





```
/// salt & 0x80000000000000000000000000000000 == true => 本次swap的接收者为EOA，否则为合约地址
/// salt & 0x40000000000000000000000000000000 == true => 本次swap要减免service fee，否则要收取
/// salt & 0x20000000000000000000000000000000 == true => meson.to;
/// salt & 0x10000000000000000000000000000000 == true => API;
/// salt & 0x08000000000000000000000000000000 == true => use *non-typed signing* (some wallets
such as hardware wallets don't support EIP-712v1);
/// salt & 0x04000000000000000000000000000000 == true => swap for core token (n/a for
releasing to contract);
/// salt & 0x0000ffffffffffffffffffff: customized data that can be passed to
integrated 3rd-party smart contract; 计算公式： $\text{uint64}(\text{encodedSwap} \gg 128)$ ;
```

- uint40 fee：本次swap要支付给 LPs (liquidity providers)的费用（注：和serviceFee是不同的），计算公式： $\text{fee} = (\text{encodedSwap} \gg 88) \& 0xFFFFFFFF$
- uint40 expireTs：本次swap在起始链的过期时间（The LP should `executeSwap` and receive his funds before `expireTs`）
- uint8 outToken、inToken：由Meson在`tokenForIndex`中定义的token索引
- uint16 outChain、inChain：起始、目标链的标识符（和我们常用的chainId不一样），可参考[SLIP-44](#)

Swap Id

Swap DONE

SWAP ID	0x63bed0a3695853f1b358ebbeca44426358e474fca457a31d279f42c08359de03
ENCODED AS	0x01012a05f200c0000000000002f230c7c000001adb0006543c6c302ca21003c02
FROM	 Ethereum  0xc6a16aa28b9b5ec16c91b3a937c438aef8ccfe14
TO	 BNB Chain  0xc6a16aa28b9b5ec16c91b3a937c438aef8ccfe14

- 如上图Meson Explorer所示，每一次swap都会有一个swap id，其计算公式为：

```
SwapId=keccak256(abi.encodePacked(encodedSwap, initiator))
```

重要状态变量

```
// `poolOfAuthorizedAddr[address] = i`, which means these addresses can all sign to
match (call `bondSwap`, `lock`) a swap and complete it (call `release`) with funds in
pool `i`.

mapping(address => uint40) public poolOfAuthorizedAddr;

//authorized addresses cannot withdraw funds from the LP pool, unless it's given in
`ownerOfPool`
// The pool index 0 is reserved for use by Meson
mapping(uint40 => address) public ownerOfPool;

// 保存各个pool中各token的余额信息
//键: poolTokenIndex (如下所示, 包含token信息) 值: 对应token的余额
// poolIndex=0时, 保存的是Meson收取的各种service fee
mapping(uint48 => uint256) internal _balanceOfPoolToken;

//tokenIndex-> poolTokenIndex的计算公式: `tokenIndex:uint8/poolIndex:uint40`
function _poolTokenIndexFrom(uint8 tokenIndex, uint40 poolIndex) internal pure returns
(uint48) {
    return (uint48(tokenIndex) << 40) | poolIndex;
}
// 可以反向推导出:
//poolTokenIndex->tokenIndex的计算公式
function _tokenIndexFrom(uint48 poolTokenIndex) internal pure returns (uint8) {
    return uint8(poolTokenIndex >> 40);
}
//poolTokenIndex->poolIndex的计算公式
function _poolIndexFrom(uint48 poolTokenIndex) internal pure returns (uint40) {
    return uint40(poolTokenIndex);
}

//键: encodedSwap
//值: postedSwap = swap initiator:address/poolIndex:uint40
```

```
mapping(uint256 => uint200) internal _postedSwaps;
```

postedSwap=0表示该swap请求目前不存在（之前可能存在过）

postedSwap=1表示该swap请求已完成、或已取消，不能再被executed或cancelled

postedSwap>1表示该swap请求可以继续execute、可以被cancel

重要方法

1. postSwap(uint256 encodedSwap, bytes32 r, bytes32 yParityAndS, uint200 postingValue):

- 参数：
 - encodedSwap: 编码后的swap信息，同时会作为 `_postedSwaps` 变量的键
 - r、yParityAndS为swap initiator（本次swap发起者、不一定是本次交易的发起者）的签名信息
 - postingValue: 作为 `_postedSwaps` 变量的值
- 本方法的作用：向Meson合约**提交swap请求**，同时将swap in token转入合约中。合约中会将swap信息记录到 `_postedSwaps` 中
- 注：任何普通用户都可以直接调用此合约方法提交swap请求，或者将swap请求及签名信息链下提交给Relayer、由Relayer代为调用此方法（两者的区别是，普通用户调用 `postSwap()` 时，postingValue中的poolIndex=0、表示不指定；Relayer调用 `postSwap()` 时，需要指定postingValue中的poolIndex，合约中会校验poolOfAuthorizedAddr来判断该Relayer是否有权限）

2. postSwapFromInitiator(uint256 encodedSwap, uint200 postingValue)

- 普通用户可以直接调用此合约方法提交swap请求，是1.的一种特殊情况。此时交易发起者msg.sender必须是swap initiator，因此参数中无需再指定签名信息

3. postSwapFromContract(uint256 encodedSwap, uint200 postingValue, address contractAddress)

- 交易发起者msg.sender是一个合约（与参数contractAddress一致），是1.的一种特殊情况。此时，发起者合约必须实现 `IAuthorizer` 接口

4. bondSwap(uint256 encodedSwap, uint40 poolIndex)

- 参数：
 - encodedSwap: 编码后的swap信息，同时会作为查找 `_postedSwaps` 变量的键
 - poolIndex: 作为 `_postedSwaps` 变量的值的一部分
- 本方法的作用：仅poolIndex对应的pool owner 或者authorized addresses可以调用，来将用户的swap请求绑定到自己的pool上。如果用户的swap请求是用户自己通过 `postSwapXXX()` 提交的，则需要由Relayer来调用此方法。如果一开始就是Relayer通过 `postSwapXXX()` 提交的、则无需再次调用此方法，因为在 `postSwapXXX()` 中已经补全了poolIndex信息

5. cancelSwap(uint256 encodedSwap)

- 作用：根据encodedSwap、取消**之前提交的、已过期未执行的**swap请求（核心原理：删除_postedSwaps[encodedSwap]），同时将之前提交请求时转入Meson合约的token再还给swap initiator。（所有人都是可以调用此方法？？只要指定了encodedSwap、就可以帮其他人取消swap？是否会有恶意取消的情况）

6. cancelSwapTo(uint256 encodedSwap, address recipient, bytes32 r, bytes32 yParityAndS)

- 参数：r、yParityAndS是encodedSwap对应的swap initiator，关于摘要“(encodedSwap, recipient)”的签名信息，证明其同意将资金退还给recipient
- 作用同5.。主要区别是退还token时，接收者不是swap initiator，而是参数中指定的recipient。为了保证资金安全，需要swap initiator的签名授权信息

7. executeSwap(uint256 encodedSwap, bytes32 r, bytes32 yParityAndS, address recipient, bool depositToPool)

- 参数：
 - r、yParityAndS是encodedSwap对应的swap initiator提供的release signature
 - recipient:
 - depositToPool: true表示资金继续保存在Meson合约中，更新pool对应的_balanceOfPoolToken余额；false表示资金转给pool owner
- 本方法的作用：swap initiator在目标链接收到资产后，调用起始链的该方法后，用户存入Meson合约的起始资产将会转给执行本次跨链操作的pool(记录到_balanceOfPoolToken 或者直接将资产转给pool owner)
- 所有人都是可以调用此方法

8. directExecuteSwap(uint256 encodedSwap, bytes32 r, bytes32 yParityAndS, address initiator, address recipient)

- 将postSwap、bondSwap、executeSwap三个操作合并到一起执行。与7.相比，主要区别是，无需校验该swap请求是否已存在。
- 主要内容：swap initiator向合约存入token、校验release signature后将swap in资产记录到bounded pool的余额中
- 调用此方法时，需要用户一次性提供request Signature、releaseSignature（Meson API采用的就是此种方法）
- 【目标链】上，搭配调用directRelease()

9. simpleExecuteSwap(uint256 encodedSwap)

- 作用同8.，主要区别是：交易发起者就是swap initiator，因此无需再提供签名信息

10. directSwap(uint256 encodedSwap, address recipient, bytes32 r, bytes32 yParityAndS)

- 参数：
 - encodedSwap: 同上

- recipient: swap out资产的接收者
- r、yParityAndS: swap发起者对swap请求的签名,
- 本方法的作用: 非跨链swap, 直接在当前交易内完成**当前链内兑换**
- 注: 目前只有premiumManager才可以使用该功能。

如何查看Meson Explorer

- 下图为Meson Explorer的首页, 展示了最新提交的各swap请求信息, 各字段的含义如下:
 - swap id/time: swapId的计算公式可参考上一模块; time为请求提交时间
 - status: 取值可参考上方“swap请求的各种状态”模块
 - from: swap initiator, 即swap用户
 - to: 本次swap的目标链的接收者
 - amout: swap数量及币种信息
 - fee: Service fee 和LP fee的总和
 - duration: swap请求从提交到Meson (relayer上链之后才会变为posted状态) 到released状态的时长

meson explorer						
Search by swap id, encoded or address						
Latest Swaps						
SWAP ID / TIME	STATUS	FROM	TO	AMOUNT	FEE	DURATION
0x527f00...eddfac 2023/11/7 16:29:22	DONE	0x5311f...d6713d zkSync Era	0x5311f...d6713d Scroll	0.0008 ETH → ETH	0.000682 ETH	00:34
0xaa7130...d64942 2023/11/7 16:29:13	BONDED	0x504c47...22c01a Linea	0x504c47...22c01a Scroll	446.060931 USDC → USDC	0.926061 USDC	-
0x14aba5...422e0c 2023/11/7 16:26:38	DONE	0x985ee0...7e7987 Polygon PoS	0x985ee0...7e7987 Mantle	1276.171982 USDT → USDT	0.0 USDT	00:24
0x12f062...a0b248 2023/11/7 16:25:21	DONE	0xfc99f5...99ee9f BNB Chain	0x0c557f...b9639f Polygon PoS	1000.0 USDT → USDT	0.53 USDT	01:00
0x7ad143...82bfac 2023/11/7 16:24:22	DONE	0x1fbf4b...a018a0 Mantle	0x1fbf4b...a018a0 zkSync Era	15.115039 USDT → USDC	0.902673 USDC	01:46
0xe64039...97edf6 2023/11/7 16:24:19	DONE	0x73b9c5...6960de BNB Chain	0x73b9c5...6960de Polygon PoS	10.0 USDT → USDC.e	0.531 USDC.e	01:44
0xd7130e...9c4d5c 2023/11/7 16:20:39	DONE	0x806d70...84bf3b Ethereum	0x806d70...84bf3b Scroll	0.00484 ETH → ETH	0.000681 ETH	02:14
0xc71572...a7de7e 2023/11/7 16:18:41	DONE	0x504c47...22c01a Arbitrum	0x504c47...22c01a Linea	447.621017 USDT → USDT	1.56 USDT	01:17

- 点击某一个“swap id”链接后，可以查看该swap的详细信息，具体可参考下图中的注释

meson explorer

Swap DONE

SWAP ID

0x94465e965c38bd4fea2f0b107dfc996b78112d8e6651dde5348b538ae36dfbe

ENCODED AS

0x01000112a88080000000000e38c71cf00006e4e4800654a0c1e003c02026601

FROM

Optimism

0x8a687ae1811c8dc4670f9331caf51330e2b7ae86

TO

Ethereum

0x8a687ae1811c8dc4670f9331caf51330e2b7ae86

AMOUNT

18.0 USDC.e → 10.271 USDT

FEE

7.729 USDT
0.5 Service fee + 7.229 LP fee

Meson后台接收到swap请求的时间（和bonded交易时间有些许差距）
REQUESTED AT

2023/11/7 16:36:42

FINISHED AT

同released交易上链的时间
2023/11/7 16:38:35

DURATION

01:52

PROCESS

Request by

swap initiator
0x8a687ae1811c8dc4670f9331caf51330e2b7ae86

Bonded

【起始链】LP将自己的pool 绑定到用户请求上
0x27e07395304ae2bc2feac2f6e8a4f6e6a0c7a243576214e638731adb2d12fd1f

Locked

【目标链】LP锁定swap out所需token
0xa3cb786bec3f0c3f5400291df6d22372e52ce529de7801fcd62f8c0bcd6ca133

Release to

0x8a687ae1811c8dc4670f9331caf51330e2b7ae86

Executed

【起始链】释放swap in token给LP
0xd812df913bf4e41e7c8d630ea821fa55e8ff31d7c99ecf7b5c08aa28f685b390

Released

【目标链】释放swap out token给 swap接收者
0x71c2b0874f9cbbcc2adb040ae4a829cbfb56534585d3f5069db6a6d71efd1c

哈希处理

具体如何生成的，可参考“合约源码解析”模块的encodedSwap部分

整个swap过程涉及到的链上交易