周亚男 2020131062

完善合约代码

04_Exchange.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IFactory {
    function getExchange(address tokenAddress) external returns
(address);
}

contract Exchange is ERC20 {
    address public tokenAddress;
    address public usdtAddress;
    address public factoryAddress;

    // events
    event TokenPurchase(
        address indexed buyer,
        uint256 indexed usdtSold,
        uint256 tokenBought
    );
    event UsdtPurchase(
        address indexed buyer,
        uint256 indexed tokenSold,
        uint256 usdtBought
    );
    event AddLiquidity(
        address indexed provider,
        uint256 indexed usdtAmount,
        uint256 indexed tokenAmount
    );
    event RemoveLiquidity(
        address indexed provider,
        uint256 indexed usdtAmount,
        uint256 indexed tokenAmount
    );
```

```
    constructor(address token, address usdt) ERC20("cbiswap", "CBI") {
        require(token != address(0), "invalid token address");
        tokenAddress = token;
        usdtAddress = usdt;
        factoryAddress = msg.sender;
    }

    function addLiquidity(
        uint256 tokenAmount,
        uint256 usdtAmount
    ) public returns (uint256 liquidity) {
        // Retrieve reserves
        (uint256 tokenReserve, uint256 usdtReserve) = getReserves();
        if (tokenReserve == 0) {
            IERC20(tokenAddress).transferFrom(
                msg.sender,
                address(this),
                tokenAmount
            );
            IERC20(usdtAddress).transferFrom(
                msg.sender,
                address(this),
                usdtAmount
            );
            liquidity = _sqrt(tokenAmount * usdtAmount);
        } else {
            // 这一步是干啥的
            usdtReserve =
                usdtReserve -
                IERC20(usdtAddress).balanceOf(address(this));

            uint256 expectedTokenAmount =
(IERC20(usdtAddress).balanceOf(
                address(this)
            ) * tokenReserve) / usdtReserve;
            require(
                tokenAmount >= expectedTokenAmount,
                "Insufficient token amount"
            );

            IERC20(tokenAddress).transferFrom(
                msg.sender,
                address(this),
                expectedTokenAmount
```

```
            );
            IERC20(usdtAddress).transferFrom(
                msg.sender,
                address(this),
                usdtAmount
            );
            liquidity =
                (totalSupply() *
IERC20(usdtAddress).balanceOf(address(this))) /
                usdtReserve;
        }

        _mint(msg.sender, liquidity);
        // 这里是不是错了:tokenAmount => expectedTokenAmount
        // 或者是应该放在函数里
        emit AddLiquidity(msg.sender, usdtAmount, tokenAmount);
    }

    function removeLiquidity(
        uint256 liquidity
    ) public returns (uint256 usdtAmount, uint256 tokenAmount) {
        require(liquidity > 0, "Amount of liquidity cannot be 0");
        // 判断流动性是否足够
        require(balanceOf(msg.sender) >= liquidity);
        // Retrieve reserves
        (uint256 tokenReserve, uint256 usdtReserve) = getReserves();

        // calculate the amount of Token & USDT based on the ratio
        usdtAmount = (usdtReserve * liquidity) / totalSupply();
        tokenAmount = (tokenReserve * liquidity) / totalSupply();

        // reduce supply of liquidities
        _burn(msg.sender, liquidity);
        // returns USDT & Token to the liquidity provider
        IERC20(usdtAddress).transfer(msg.sender, usdtAmount);
        IERC20(tokenAddress).transfer(msg.sender, tokenAmount);
        emit RemoveLiquidity(msg.sender, usdtAmount, tokenAmount);
    }

    // 使用特定数量的 USDT 购买 Token
    function swapExactUsdtToToken(
        uint256 amountUsdtIn,
        // 这个变量的实际意义是什么
        uint256 expectedTokenAmount,
```

```
        address to
    ) public {
        // 补全
        require(amountUsdtIn > 0);
        uint256 amountTokenOut = getAmountOut(amountUsdtIn,
usdtAddress);
        require(expectedTokenAmount <= amountTokenOut);

        IERC20(usdtAddress).transferFrom(
            msg.sender,
            address(this),
            amountUsdtIn
        );
        IERC20(tokenAddress).transfer(to, amountTokenOut);

        emit TokenPurchase(to, amountUsdtIn, amountTokenOut);
    }

    // 使用 USDT 购买特定数量的 Token
    function swapUsdtToExactToken(
        uint256 amountTokenOut,
        uint256 maxUsdtAmountIn,
        address to
    ) public {
        // 补全
        require(amountTokenOut > 0);
        uint256 amountUsdtIn = getAmountIn(amountTokenOut,
tokenAddress);
        require(maxUsdtAmountIn >= amountUsdtIn);

        IERC20(usdtAddress).transferFrom(
            msg.sender,
            address(this),
            amountUsdtIn
        );
        IERC20(tokenAddress).transfer(to, amountTokenOut);

        emit TokenPurchase(to, amountUsdtIn, amountTokenOut);
    }

    // 使用特定数量的 Token 购买 USDT
    function swapExactTokenToUsdt(
        uint256 amountTokenIn,
        uint256 expectedUsdtAmount,
```

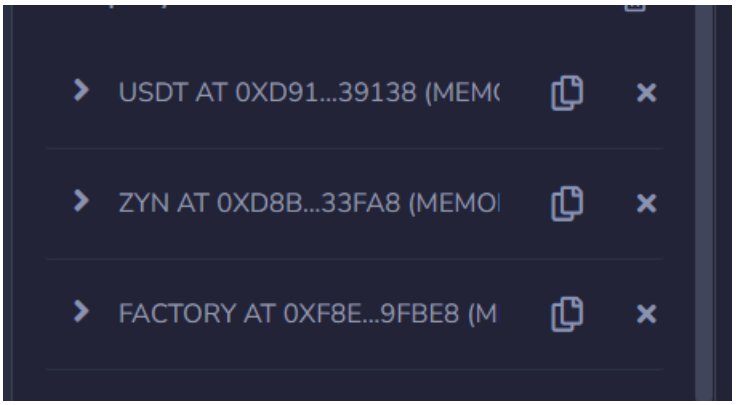
```
        address to
    ) public {
        // 补全
        require(amountTokenIn > 0);
        uint256 amountUsdtOut = getAmountOut(amountTokenIn,
tokenAddress);
        require(expectedUsdtAmount <= amountUsdtOut);

        IERC20(tokenAddress).transferFrom(
            msg.sender,
            address(this),
            amountTokenIn
        );
        IERC20(usdtAddress).transfer(to, amountUsdtOut);

        emit UsdtPurchase(to, amountUsdtOut, amountTokenIn);
    }

    // 使用 Token 购买特定数量的 USDT
    function swapTokenToExactUsdt(
        uint256 amountUsdtOut,
        uint256 maxTokenAmountIn,
        address to
    ) public {
        // 补全
        require(amountUsdtOut > 0);
        uint256 amountTokenIn = getAmountIn(amountUsdtOut, usdtAddress);
        require(maxTokenAmountIn >= amountTokenIn);

        IERC20(tokenAddress).transferFrom(
            msg.sender,
            address(this),
            amountTokenIn
        );
        IERC20(usdtAddress).transfer(to, amountUsdtOut);

        emit UsdtPurchase(to, amountUsdtOut, amountTokenIn);
    }

    function getReserves()
        public
        view
        returns (uint256 tokenReserve, uint256 usdtReserve)
    {
```

```
        tokenReserve = IERC20(tokenAddress).balanceOf(address(this));
        usdtReserve = IERC20(usdtAddress).balanceOf(address(this));
    }

    // 已知确定的输入数量和币种，计算输出数量
    function getAmountOut(
        uint256 inputAmount,
        address inputToken
    ) public view returns (uint256 outputAmount) {
        // 补全
        require(inputAmount > 0);
        require(inputToken == usdtAddress || inputToken ==
tokenAddress);
        (uint256 tokenReserve, uint256 usdtReserve) = getReserves();
        if (inputToken == usdtAddress) {
            outputAmount =
                tokenReserve -
                (usdtReserve * tokenReserve) /
                (usdtReserve + inputAmount);
        } else {
            outputAmount =
                usdtReserve -
                (usdtReserve * tokenReserve) /
                (tokenReserve + inputAmount);
        }
    }

    // 已知确定的输出数量和币种，计算输入数量
    function getAmountIn(
        uint256 outputAmount,
        address outputToken
    ) public view returns (uint256 inputAmount) {
        // 补全
        require(outputAmount > 0, "not enough");
        require(outputToken == usdtAddress || outputToken ==
tokenAddress);
        (uint256 tokenReserve, uint256 usdtReserve) = getReserves();
        if (outputToken == usdtAddress) {
            inputAmount =
                (usdtReserve * tokenReserve) /
                (usdtReserve - outputAmount) -
                tokenReserve;
        } else {
            inputAmount =
```

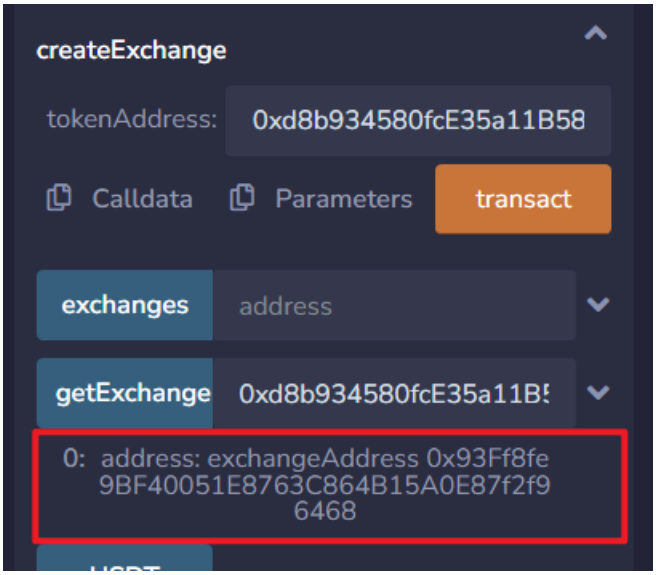
```
                (usdtReserve * tokenReserve) /
                (tokenReserve - outputAmount) -
                usdtReserve;
        }
    }

    function _sqrt(uint y) private pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}
```

实验过程

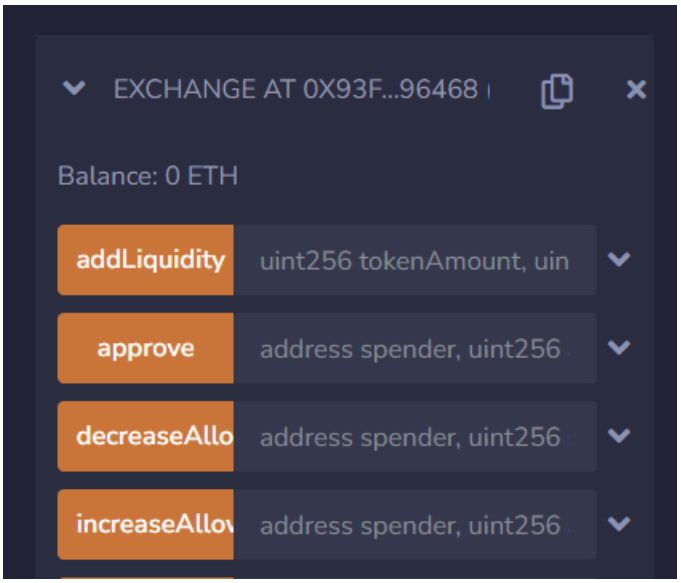
1. 仔细阅读 Factory.sol 合约，思考它与 Exchange.sol 的关系
   factory 合约记录哪些 token 和 usdt 可以交换，创建交易对
   exchange 合约是专实现交换功能的合约
2. 部署合约



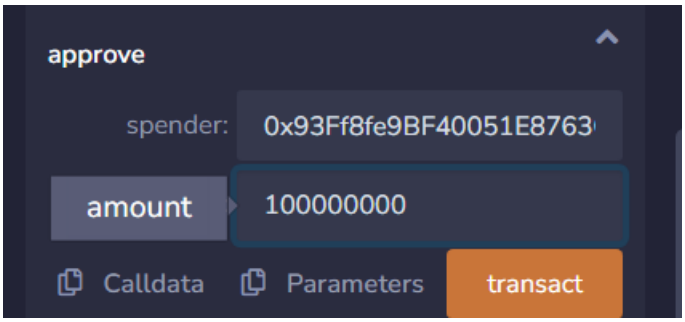
3. 调用 factory 的 createExchange 方法，创建自己名字命名的币的交易对
   查询交易对的地址

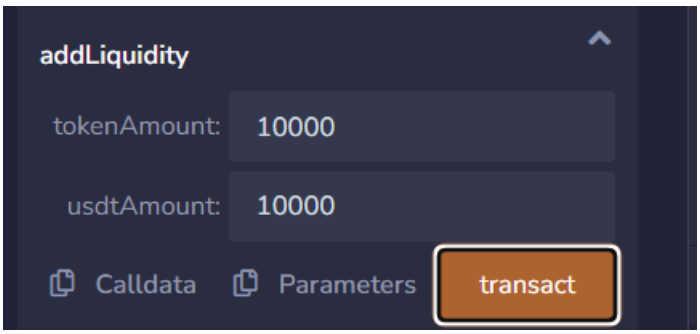


4. 引入交易对

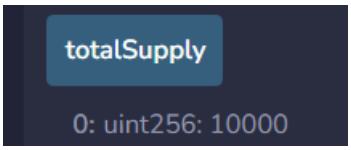


5. 添加流动性，每边 10000 个 token

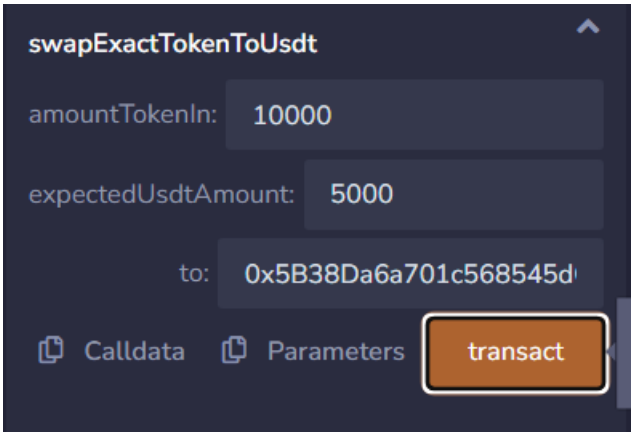




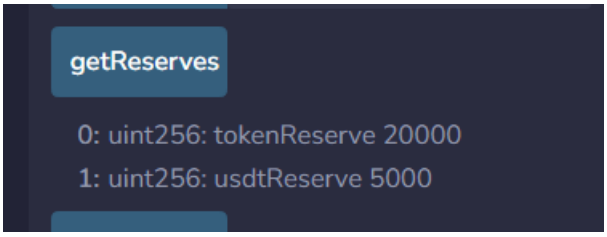
6. 查看 totalSupply



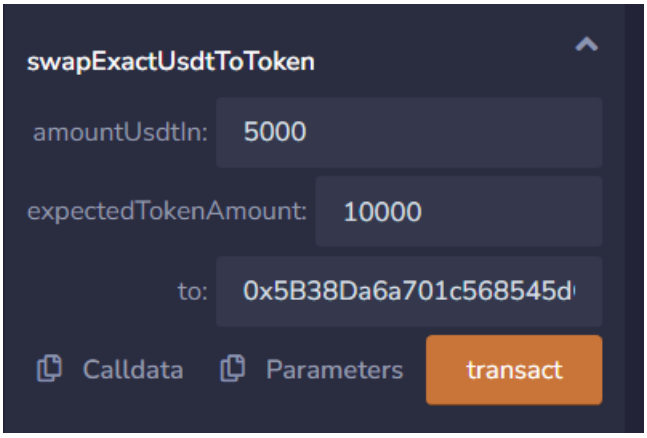
7. 调用 swapExactTokenToUsdt



8. 查看 getReserves

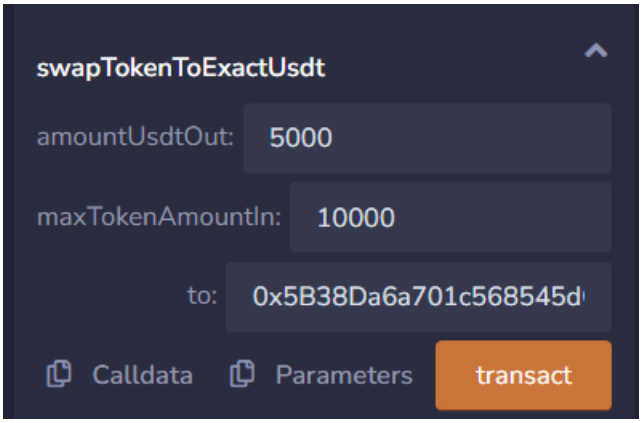


9. 调用 swapExactUsdtToToken

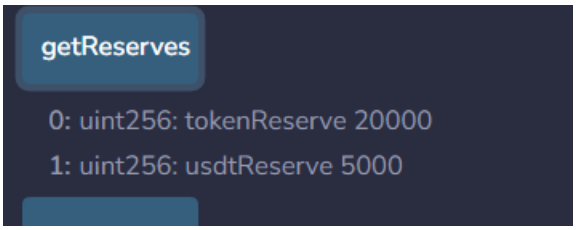


10. 查看 getReserves

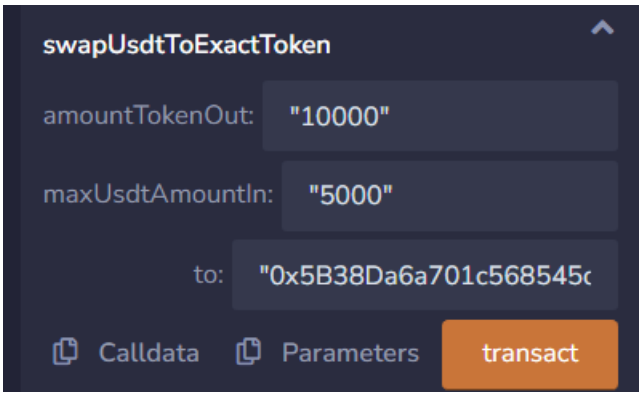
11. 调用 swapTokenToExactUsdt



12. 查看 getReserves



13. 调用 swapUsdtToExactToken



14. 查看 getReserves