# 记录hardhat踩坑

1.合约实例.address已不能获取到合约地址，得改用合约实例.target

2.在测试代码中，若部署的合约涉及到引用库合约，则先部署库合约，然后导入到部署主合约上 例子如下：

const CSHARE = await hre.ethers.getContractFactory("Contract",{

            libraries:{

                Libname:lib实例.target,

            },

        }

3.另外，在没有配置hardhat运行环境，即hre时，调用如ethers.getContractFactory

时则要加上hre. 若不加上，则会导致找不到

1. loadFixture的使用：

第一次调用 loadFixture 时，将执行fixture。但第二次时，loadFixture 不会再次执行fixture，而是将网络状态重置为执行fixture后的状态。这更快，并且它会撤消先前测试所做的任何状态更改。

可代替掉beforeEach，在每一次测试时不用再次部署合约。 另外如果想重新要一个deployment，则可以不用loadFixture，可以手动部署，例子如下：

    it("Should fail if the unlockTime is not in the future", async function() {

      // We don't use the fixture here because we want a different deployment

      const latestTime = await time.latest();

      const Lock = await ethers.getContractFactory("Lock");

      await expect(Lock.deploy(latestTime, { value:1 })).to.be.revertedWith(

        "Unlock time should be in the future"

      );

});

1. 在hardhat test脚本文件里，若调用合约的函数，函数有返回值且是view修饰符，则可以直接定义一个变量来接收返回值：

合约：function owner() public view returns (address) {

return \_owner;

}

脚本：const owner = await myNFT.owner();

如果不是view修饰，则不能直接定义一个变量接收返回值，如：

函数： function mintNFT(address recipient, string memory tokenURI)

public returns (uint256)

{

require(msg.sender == owner(), "Only owner is allowed to mint");

uint newItemId = ++\_counter;

ERC721.\_mint(recipient, newItemId);

ERC721URIStorage.\_setTokenURI(newItemId, tokenURI);

return newItemId;

}

这样的函数则按上述方法调用会报错，网上解释：You can not directly receive a return value coming from a function that you are sending a transaction to off-chain. You can only do so on-chain - i.e. when one SC function invokes another SC function.

解决办法：再写个合约，自定义变量来接收该函数返回值，即在本合约里导如函数所在合约，创建实例并且调用函数

在写测试脚本的时候，若合约有两个函数同名，但通过传入参数类型不同or数量不同来区分函数的，在hardhat里行不通。此问题仍未解决..........之后补充

时隔三天，已解决：

**解决办法，注册abi：**

如果遇到两个safeTransferFrom函数，但参数类型或个数不同

错误方式contract.safeTransferFrom(addr1, addr2, 1);

正确方式：contract["safeTransferFrom(address,address,uint256)"](addr1, addr2, 1);

**Hardhat合约测试部署可升级合约：**

const { AddressZero } = ethers.constants;

let deployResult = await deploy('', {

        contract: 'ParamcontractName',

        from: 从哪个地址部署,

        gasLimit: 30000000,

        args: [],

        proxy: {

            proxyContract: 'OpenZeppelinTransparentProxy',

            execute: {

                init: {

                    methodName: 'initialize',

                    args: [1, AddressZero]

                }

            }

        },

        log: true,

    });

const contract = await ethers.getContract('合约名字')

===========================/========================================

在写hardhat的测试时，**若it这个单元测试报错超时**，则在后面限定一下

        it("", async function () {

        }).timeout(1000000)

这里可以让超时报错消失

方法2：

在hardhat-config.js文件里加上限定：

mocha: {

    timeout: 10000000

  },

**写hardhat测试文件时的calldata的打包**

以下是打包erc20转账的演示

//先把打包函数搞出来

const abiCoder = new ethers.AbiCoder();

            // 将function转换为buffer

            const functionBytes = toUtf8Bytes('transfer(address,uint256)');

            // 打包参数

            const data = abiCoder.encode(['address', 'uint256'], [user4.address, ethers.parseEther("10")]);

            // 获取函数选择器

            const functionSelector = dataSlice(keccak256(functionBytes), 0, 4);

            // 拼接calldata

            const finalData = functionSelector + data.substring(2);

最终传入finaldata即可调通

另外一种方式： 之后补充

//===========================================================

在solidity编译期间，若修改过合约名字大小写or文件夹名字大小写，再次编译可能报错：

Error HH702: Invalid artifact

解决办法：

npx hardhat clean，再次npx hardhat compile

//---------------------------------------------------------------

在运行测试时用npx mocha test/test.js 报错没有etherjs

那么用npx hardhat test即可

//=====================================================================

与已经部署的合约进行交互：

如果您正在使用附件，下面是代码参考

let xyzContract = await hre.ethers.getContractFactory("Name of the contract");

let xyzContractInstance = xyzContract.attach('Address of the contract');

当使用hardhat时，您可以通过getContractAt()在一行中完成相同的事情

let xyzContractInstance = await hre.ethers.getContractAt(

"Contract Name",

deployed contract address

);

//=============================================

用mock技术可以不用把其他项目的合约扒过来，以下是如何使用mock的示例：

在有hardhat-deployer工具的时候：

①部署

await deploy('mock合约别名', {

        contract: "Mock合约名字",

        from: deployer,

        gasLimit: 30000000,

        args: [],

        log: true,

    })

②写测试脚本的时候：

**const** 合约实例 **=** **await** ethers.**getContract**("mock合约别名")

③最重要的：

contract Mock is I自己定义的接口 {

    constructor() {}

//需要用到的函数：形式参数名字，返回参数名字都可以不写，直接写类型就行

    function fc(address, address) external view returns (address, address, address, address) {}

}

//----------------------------------------------------------------关于合约升级的插槽的处理办法

如果老版本的合约有继承其他的合约，并且还声明了其他的变量，在处理升级时，可以把之前的合约弄成一个abstract合约，把声明的变量全部弄到abstract合约里，然后再写一个合约继承abstract合约，并且继承之前老合约继承的合约。最后再去layout上查看合约插槽变换，再做微调

// =========================//=========================

## 关于使用hardhat-deploy插件的坑

npm依赖问题：

Found: hardhat-deploy@0.12.2

npm ERR! node\_modules/hardhat-deploy

npm ERR! dev hardhat-deploy@"^0.12.2" from the root project

npm ERR!

npm ERR! Could not resolve dependency:

npm ERR! peer hardhat-deploy@"^0.11.34" from @nomiclabs/hardhat-ethers@0.4.1

npm ERR! node\_modules/@nomiclabs/hardhat-ethers

npm ERR! dev @nomiclabs/hardhat-ethers@"npm:hardhat-deploy-ethers" from the root project

[这里报错是hardhat-deploy@"^0.11.34"已经有了@nomiclabs/hardhat-ethers@0.4.1](mailto:这里报错是hardhat-deploy@\"^0.11.34\"已经有了@nomiclabs/hardhat-ethers@0.4.1)

然而我使用的是hardhat-deploy@"^0.12.2"版本，所以，运行npm install --save-dev @nomiclabs/hardhat-ethers@npm:hardhat-deploy-ethers ethers时会报版本冲突的问题，依赖无法下载

解决办法：使用"hardhat-deploy": "^0.11.24"版本，再运行npm install --save-dev @nomiclabs/hardhat-ethers@npm:hardhat-deploy-ethers ethers即可。 或者下载时再后面加上--force 或 --legacy-peer-deps 强行下载，但是可能会导致依赖不能正常使用的错误

// =====================//====================

**计算传入结构体的函数的函数选择器**：

function fillOrder(

Order memory order,

uint256 takerAssetFillAmount,

bytes memory signature

)

public

returns (LibFillResults.FillResults memory fillResults);

计算方式：fillOrder((address,address,address,address,uint256,uint256,uint256,uint256,uint256,uint256,bytes,bytes),uint256,bytes)

//======================//================================

在使用gitflow时，不能直接去gitlab建立hotfix/0.xx.x分支，应该在vscode上用gitflow建立。并且，gitflow是用master分支的代码来搞的hotfix分支。另外，在建立新的hotfix分支时，要先把之前的hotfix分支finish了，finnish时需要把hotfix合并到master分支（或者直接删了）。tag在创建hotfix时会生成。

Git指令：指定克隆某分支git clone --branch <branchname> <remote-repo-url>或者git clone -b <branchname> <remote-repo-url>

Git rebase 命令：合并分支或者合并之前提交的记录