FORESIGHT X

<OpenBuild Hack@HK/>

<Solidity Strengthen Workshop />

Frank

Security Tech Lead @ Beosin

Foresight X
Gateway to
Web3 in Hongkong

OpenBuild
HACK.
SERIES

BLOCKCHAIN ACADEMY × BEOSIN Blockchain Security

Hardhat 教程

Contents

01 Hardhat 介绍

简介

一个灵活、快速、可扩展的以太坊智能合约开发环境

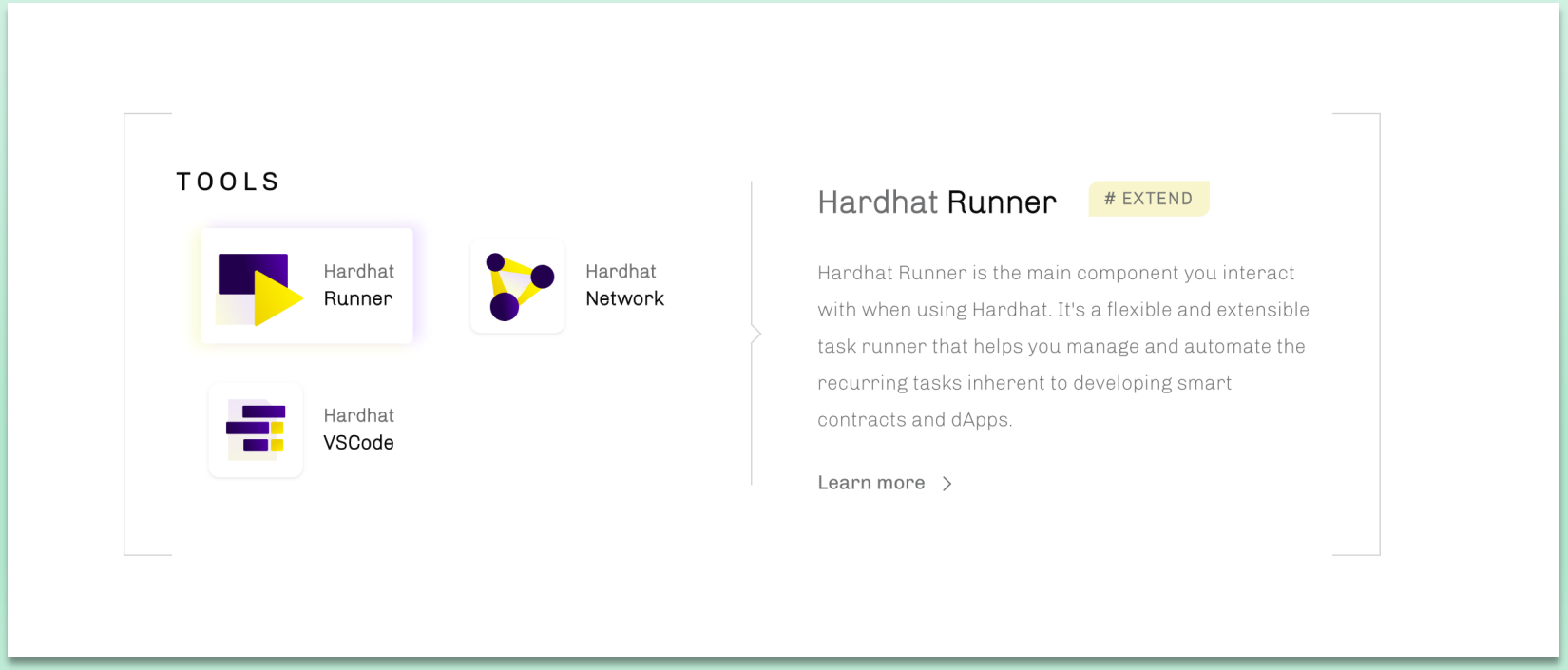
Flexible. Extensible. Fast.

Ethereum development environment for professionals

Hardhat组成

Hardhat共有三个组成部分：

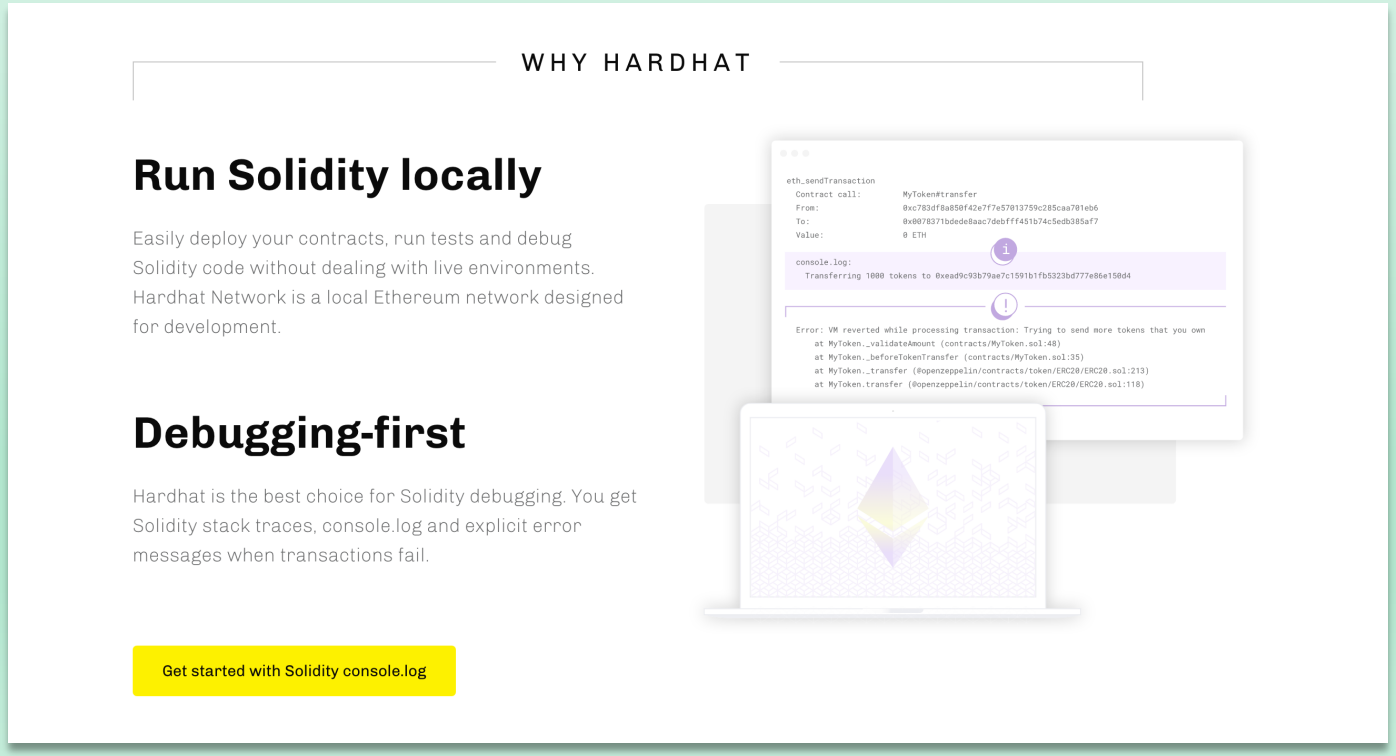
- Hardhat Runner：Hardhat运行环境，核心为task和plugin
- Hardhat Network：开发使用的本地ETH网络
- Hardhat VSCode：为VSCode中开发Solidity设计的插件



Hardhat便捷性

Hardhat便捷性有多个方面：

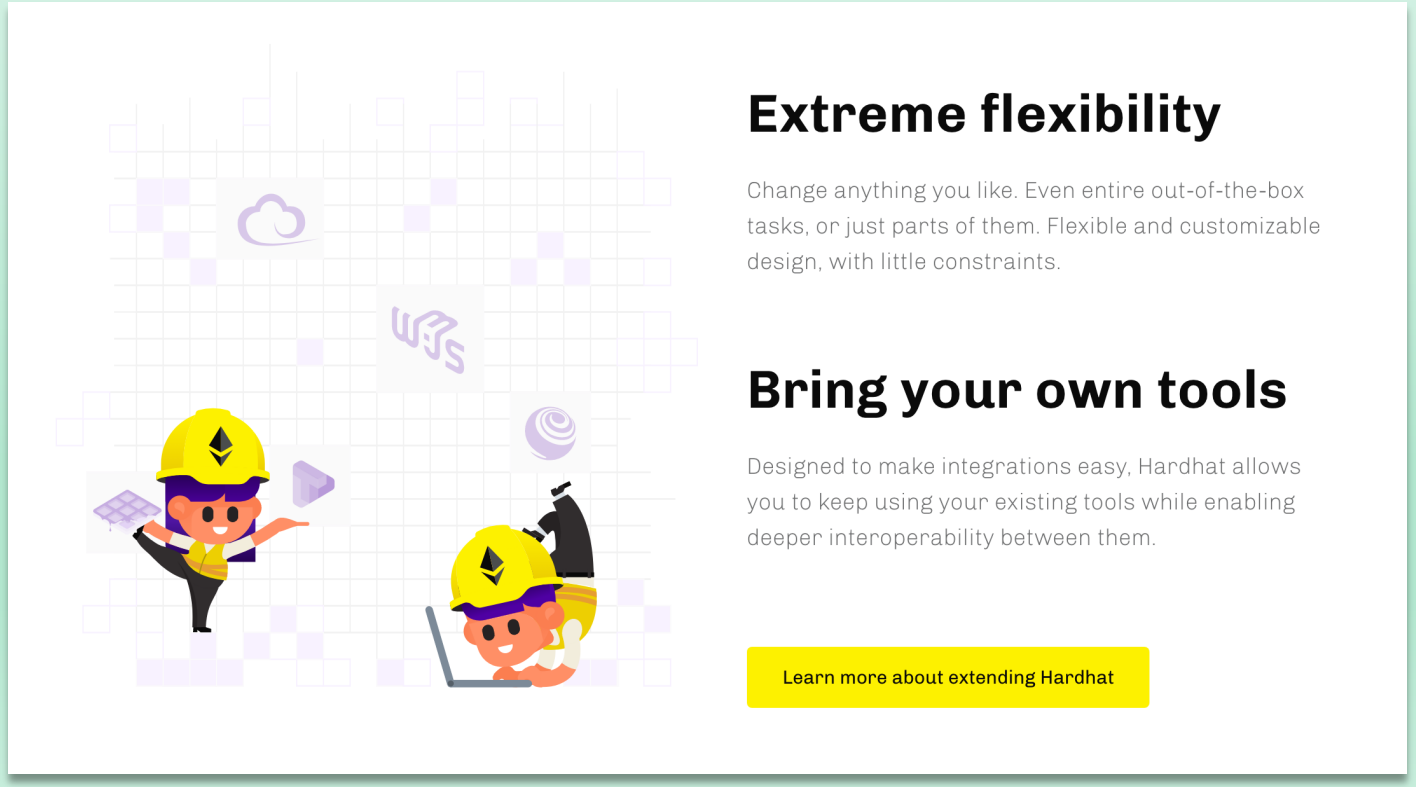
- 无需配置网络环境，在本地开发智能合约
- 利用Hardhat Network模拟部署和测试
- 丰富的Debug工具，包括合约中使用console.log，以及返回详细的错误信息和堆栈追踪信息



Hardhat优点

除了便捷性，还有多个优点：

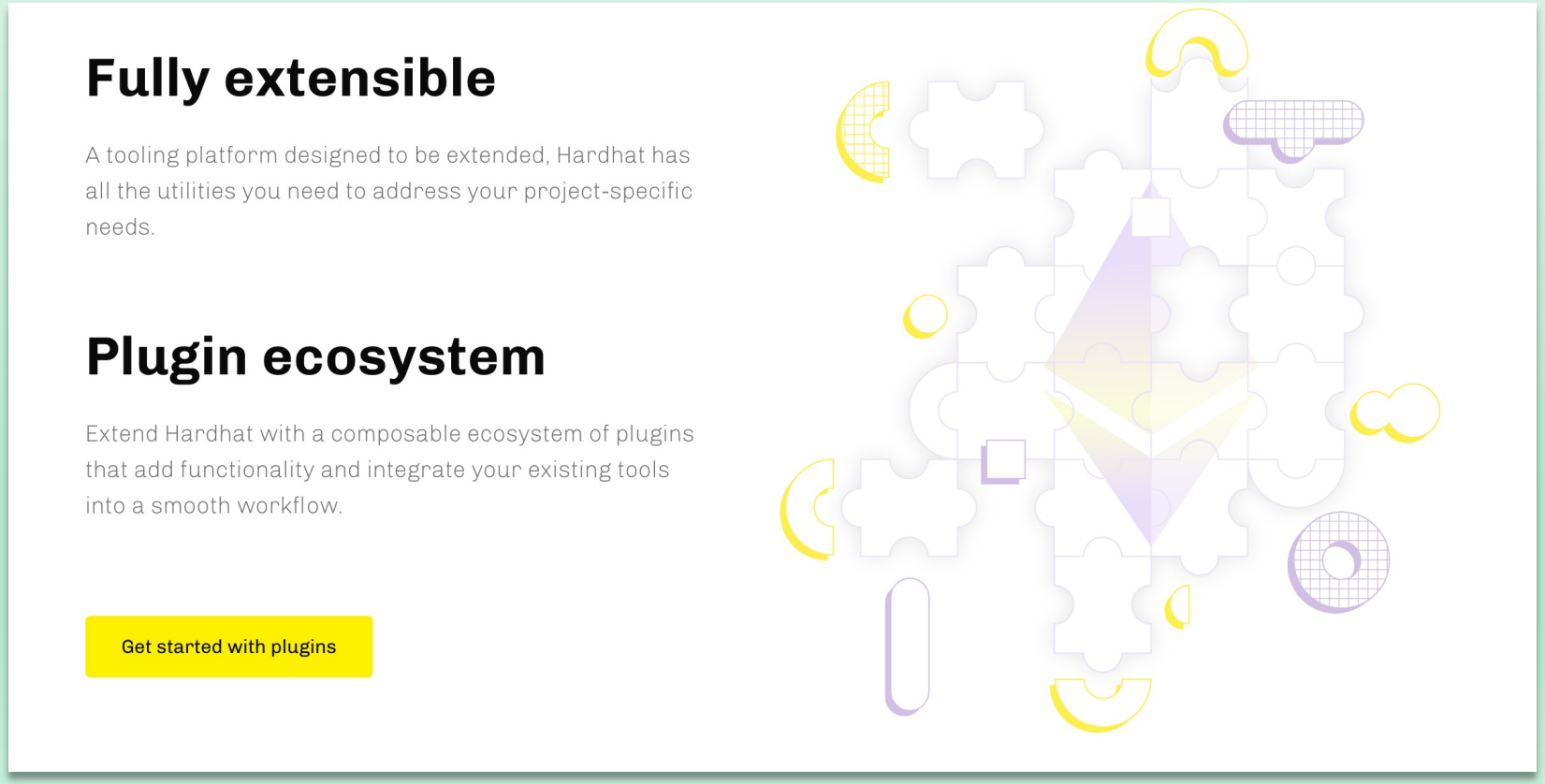
- 灵活性：自由地配置项目并构建自己的task
- 兼容性：可以和其他开发工具一起使用(Truffle, Foundry, DappTools)



Hardhat优点

除了便捷性，还有多个优点：

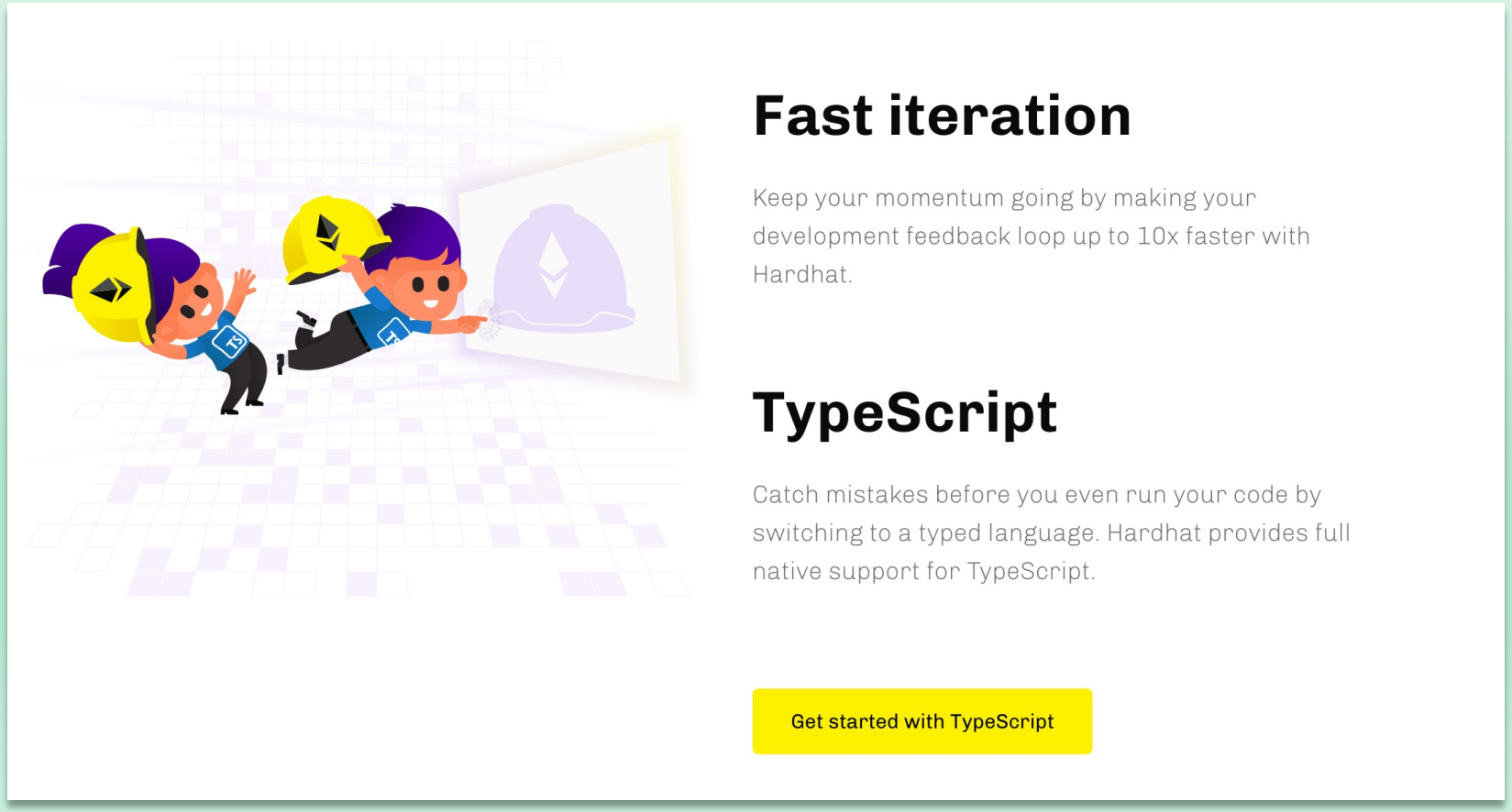
- 扩展性：可以添加各种插件来扩展功能
- 插件：拥有丰富的插件生态，活跃的社区支持



Hardhat优点

除了便捷性，还有多个优点：

- 快速迭代：更新周期快，社区反馈导向
- 原生支持TypeScript，更方便、清晰和规范地书写测试、脚本和任务



Remix和Hardhat对比

Remix优点：

- 在线编辑，简单方便
- 支持合约部署和交互
- 支持测试和脚本

Hardhat优点：

- 本地开发，可以使用更强大的代码编辑器及各种插件
- 工程化、批量化的测试和脚本
- 定制化的Task
- 丰富的社区插件

02 Hardhat基本配置

准备工作

安装NodeJS

- Hardhat是基于NodeJS的框架

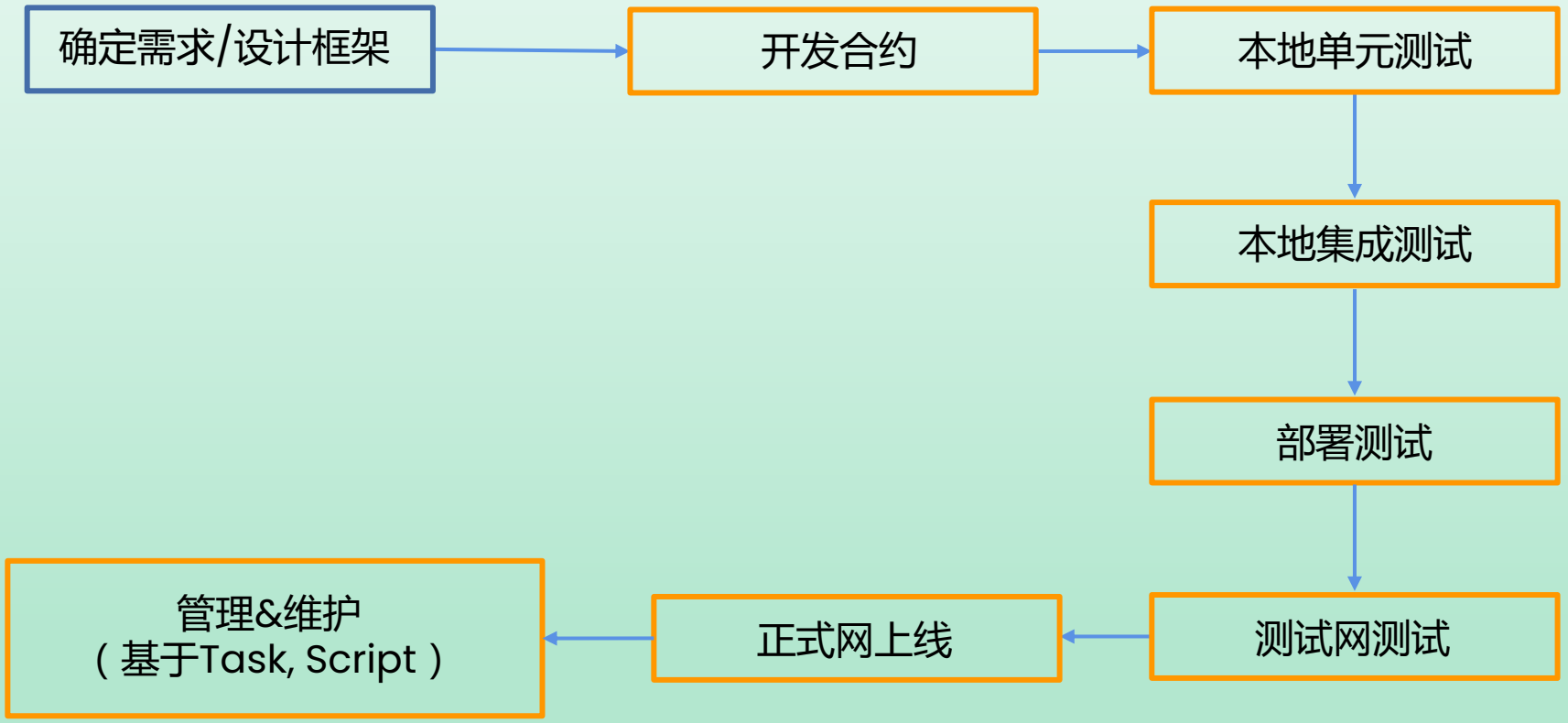
选择你的代码编辑器

- 推荐VSCode
- 插件安装：Solidity

准备必要的Key和资产

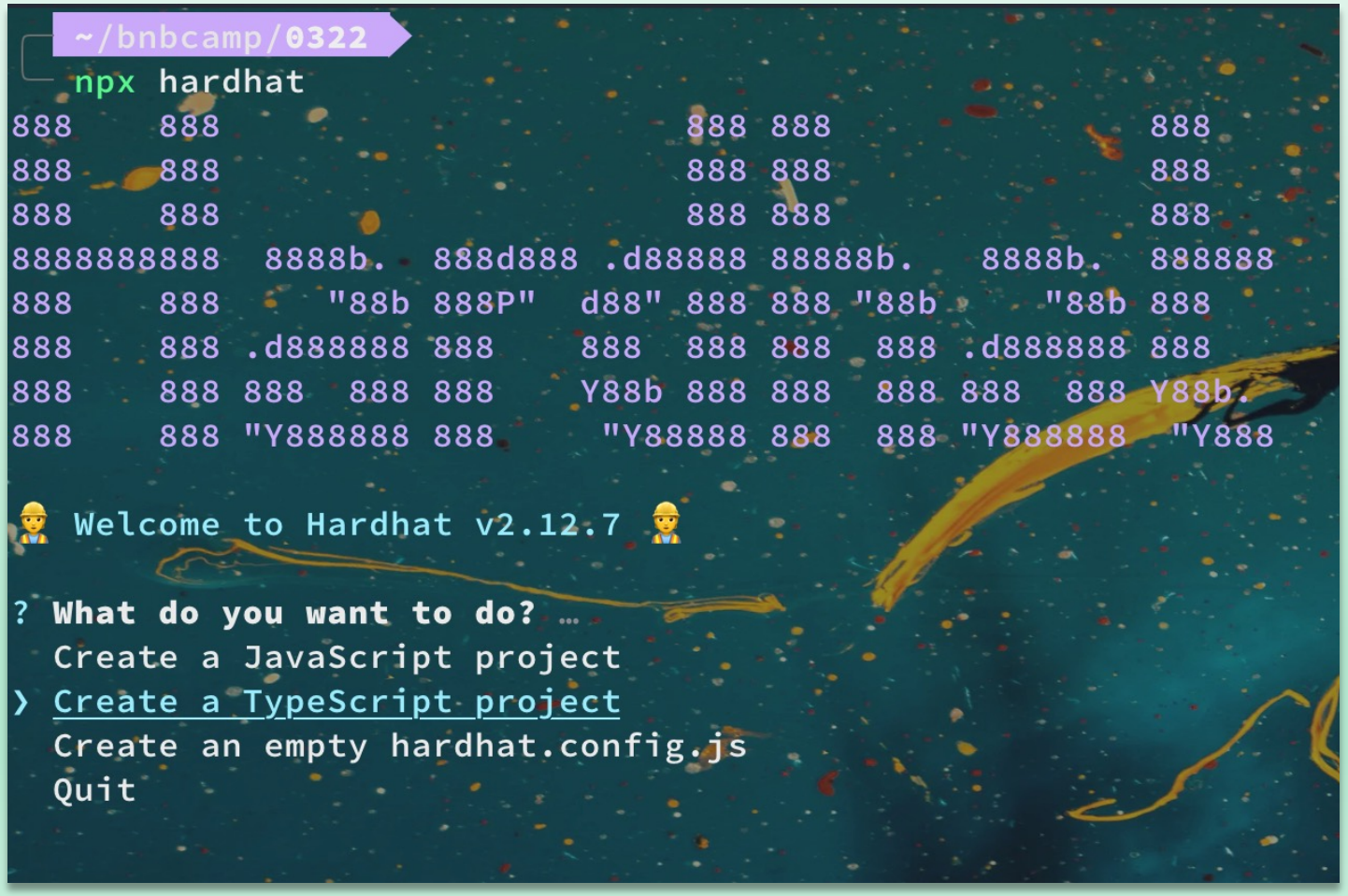
- 注册BSCScan的APIKey用于验证合约代码
- 领取BNB Testnet的测试token
- 注册Coinmarketcap的APIKey用于测试中显示gasFee(可选)

Hardhat工程化开发流程



Hardhat项目创建步骤

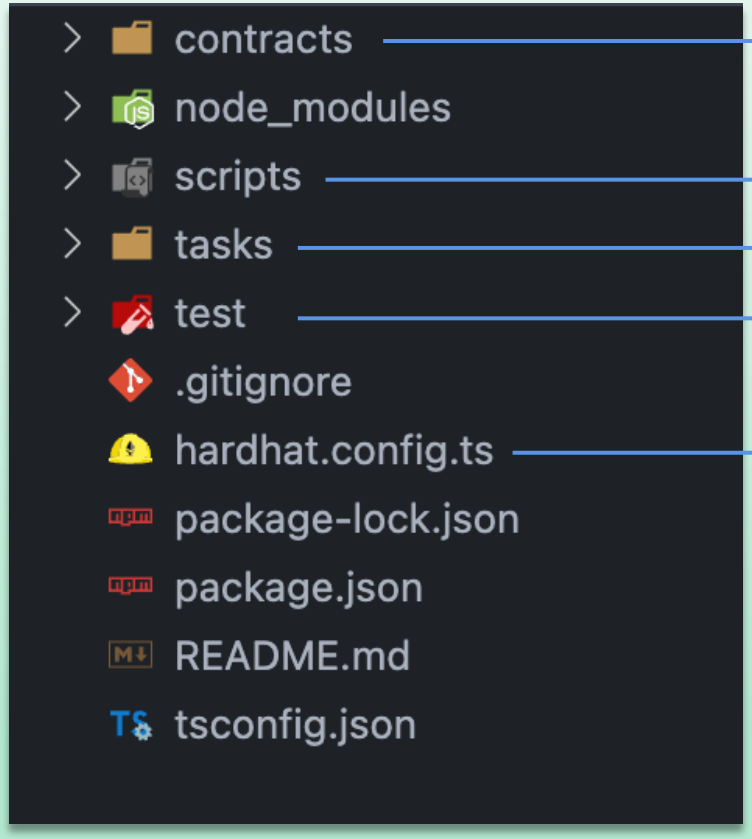
$ npx hardhat



- 选择TypeScript项目
- 安装hardhat/toolbox

  @nomicfoundation/hardhat-toolbox
- 安装dotenv使用环境变量

Hardhat目录结构



合约文件

脚本文件

任务

测试

Hardhat配置文件

Hardhat目录结构

```
1  solidity: {
2    version: "0.8.19",
3    settings: {
4      optimizer: {
5        enabled: true,
6        runs: 200,
7      },
8    },
9  },
10 paths: {
11   sources: "./contracts",
12   tests: "./test",
13   cache: "./cache",
14   artifacts: "./artifacts",
15 },
16
```

Solidity编译器设置

- 可以选择单个版本或多个版本
- 可以为某个文件单独设置编译器选项

项目路径设置

Hardhat配置文件

```
1  defaultNetwork: "hardhat",
2  networks: {
3    hardhat: {},
4    localhost: {
5      url: "http://127.0.0.1:8545",
6    },
7    bnbtest: {
8      url: process.env.BNBTest_URL,
9      accounts: {
10       mnemonic: process.env.BNBTest_MNEMONIC,
11       count: 10,                    助记词
12     },
13   },
14   bnb: {
15     url: process.env.BNB_URL,
16     accounts:
17       process.env.BNB_PRIVATE_KEY !== undefined
18         ? [process.env.BNB_PRIVATE_KEY]
19         : [],                       私钥
20   },
21 },
```

默认网络

网络配置

- url：连接的RPC节点的URL
- accounts: 在该网络使用的账户(用于发起交易、签名等，可以使用私钥或助记词来配置)

常用网络

- hardhat：默认网络，无状态，只可验证部署流程
- localhost：一般配置为hardhat本地启动的网络(使用npx hardhat node启动)
- bnbtest：测试网
- bnb：主网

03 Hardhat部署与测试

Hardhat的Debug工具

使用console.log来帮助开发和debug

```sol
import "hardhat/console.sol";

contract Console {
    uint256 public value;

    function setValue(uint256 _newValue) external {
        console.log(_newValue);
        console.log("new value is ", _newValue);
        console.log("doubled new value is ", _newValue * 2);

        value = _newValue;
    }
}
```

- 可以打印多种类型数据
- 不会影响实际部署的代码

Hardhat部署与测试

Owner机制

- 合约的部署者成为owner，拥有权限

白名单机制

- 仅在白名单上的地址可以mint
- Owner可以添加白名单
- 每个地址只能mint一个

转移机制

- 不可转移

Burn机制

- 只有合约的owner可以burn掉任意地址的NFT

准备

- 安装@openzeppelin/contracts

  $ npm install @openzeppelin/contracts

- 引入并继承ERC721合约

```
1
2 import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
3 contract MyPunks is ERC721 {
4
```

准备

- 实现owner权限修饰器

```
1
2  address public owner;
3
4  constructor() ERC721("MyPunks", "PUNK") {
5      owner = msg.sender;  部署者成为owner
6  }
7
8  modifier onlyOwner() {
9      require(msg.sender == owner, "Not owner");
10     _;      限制仅owner调用
11 }
12
```

准备

- 实现白名单功能

```
1
2  mapping(address ⇒ uint256) public isWhitelisted;
3
4  event NewWhitelistAdded(address user);
5
6  function addWhitelist(address _user) external onlyOwner {
7      isWhitelisted[_user] = 1;
8
9      emit NewWhitelistAdded(_user);
10 }
```

添加NFT白名单

```
1
2  modifier onlyWhitelisted() {
3      require(isWhitelisted[msg.sender] == 1, "Not whitelisted");
4      _;
5  }
6
```

限制仅白名单上的用户可操作

准备

- Mint & Burn相关函数

```
1
2  function mint() external onlyWhitelisted {
3      require(userTokenId[msg.sender] == 0, "Already minted");
4
5      uint256 tokenId = ++counter;
6
7      _safeMint(msg.sender, tokenId);
8
9      userTokenId[msg.sender] = tokenId;
10
11     emit NewPunkMinted(msg.sender, tokenId);
12 }
13
14 function burn(address _user) external onlyOwner {
15     uint256 tokenId = userTokenId[_user];
16
17     _burn(tokenId);
18
19     emit PunkBurned(_user, tokenId);
20 }
21
```

准备

- 限制0x0地址转账

```
1
2     function _beforeTokenTransfer(
3         address _from,
4         address _to,
5         uint256,
6         uint256
7     ) internal pure override {
8         require(
9             _from == address(0) || _to == address(0),
10            "Transfer not allowed"
11        );
12    }
13
```

部署

- 部署到localhost

```
import { ethers } from "hardhat";

async function main() {
  const MyPunks = await ethers.getContractFactory("MyPunks");
  const mypunks = await MyPunks.deploy();

  await mypunks.deployed();

  console.log(
    `Deployed MyPunks to: ${mypunks.address} with ${mypunks.deployTransaction.hash}`
  );
}

// We recommend this pattern to be able to use async/await everywhere
// and properly handle errors.
main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});

```

部署

- 使用tasks部署到localhost，需要在hardhat.config.ts中导入tasks

$ npx hardhat deploy --network localhost

```
task("deploy", "Deploy MyPunks contract").setAction(async (_, hre) => {
  const { network } = hre;
  const [dev_account] = await hre.ethers.getSigners();

  console.log(
    "Deploying mypunks contract to network ",
    network.name,
    " by ",
    dev_account.address
  );

  const MyPunks = await hre.ethers.getContractFactory("MyPunks");
  const mypunks = await MyPunks.deploy();

  await mypunks.deployed();

  console.log(
    `Deployed MyPunks to: ${mypunks.address} with ${mypunks.deployTransaction.hash}`
  );
```

部署

$ npx hardhat addWhitelist –address 0xabc123 --network localhost

```
task("addWhitelist", "Add a new whitelist address")
  .addParam("address", "The address to be added")
  .setAction(async (taskArgs, hre) => {
    const [dev_account] = await hre.ethers.getSigners();

    const MYPUNKS_ADDRESS = "";
    const mypunks = new MyPunks__factory(dev_account).attach(MYPUNKS_ADDRESS);

    const tx = await mypunks.addWhitelist(taskArgs.address);
    console.log("Tx details: ", await tx.wait());
  });
```

$ npx hardhat mint --network localhost

```
task("mint", "Add a new whitelist address").setAction(async (_, hre) => {
  const [, alice] = await hre.ethers.getSigners();

  const MYPUNKS_ADDRESS = "";
  const mypunks = new MyPunks__factory(alice).attach(MYPUNKS_ADDRESS);

  const tx = await mypunks.connect(alice).mint();
  console.log("Tx details: ", await tx.wait());
});

```

测试

设计我们的测试内容

Deployment

- Correct owner
- Initial values

Mint

- Add whitelist
- Mint by address on/not on whitelist
- Only mint one by each address

Transfer

- Not allowed to transfer

Burn

- Only burn by owner

测试

测试准备：

- 测试框架: Mocha
- 断言库: Chai

常用断言：

- 判断返回值　expect (await {someFunction}).to.equal({expected value})
- 抛出事件　await (expect {someFunction}).to.emit({some event}).withArgs({event args})
- 抛出异常　await (expect {someFunction}).to.be.revertedWith({error message})

测试

测试变量定义和合约环境准备：

```
describe("Mypunks Test", function () {
  let MyPunksFactoty:ContractFactory;
  let myPunks: Contract;
  let admin: SignerWithAddress;
  let alice:SignerWithAddress;
  let bob: SignerWithAddress;

  beforeEach(async function () {
    [admin, alice, bob] = await ethers.getSigners();

    //获得 MyPunksFactoty
    MyPunksFactoty = await ethers.getContractFactory("MyPunks");

    //部署合约
    myPunks = await MyPunksFactoty.deploy();
    await myPunks.deployed()
  });
```

定义测试的合约和用到的账户

每个测试前都重新部署一次合约

测试

测试部署：

```
describe("Deployment", function () {
  it("should have the correct owner", async function () {
    expect(await mypunks.owner()).to.equal(admin.address);
  });

  it("should have the correct intial value", async function () {
    expect(await mypunks.counter()).to.equal(0);
  });
});
```

测试

测试mint：

```
describe("Mint", function () {
  it("should be able to add a whitelist address", async function () {
    await expect(mypunks.addWhitelist(alice.address))
      .to.emit(mypunks, "NewWhitelistAdded")
      .withArgs(alice.address);

    expect(await mypunks.isWhitelisted(alice.address)).to.equal(1);
  });

  it("should be able to mint a token", async function () {
    // Add whitelist
    await mypunks.addWhitelist(alice.address);
    // Mint NFT by alice
    await expect(mypunks.connect(alice).mint())
      .to.emit(mypunks, "NewPunkMinted")
      .withArgs(alice.address, 1);
    // Result check
    expect(await mypunks.counter()).to.equal(1);
    expect(await mypunks.balanceOf(alice.address)).to.equal(1);
    expect(await mypunks.userTokenId(alice.address)).to.equal(1);
  });

  it("should not be able to mint more than one token", async function () {
    await mypunks.addWhitelist(alice.address);
    await mypunks.connect(alice).mint();
    // Mint NFT by alice again
    await expect(mypunks.connect(alice).mint()).to.be.revertedWith(
      "Already minted"
    );
  });
});
```

测试

测试transfer：

```
describe("Transfer",function() {
  beforeEach(async function()  {
    await myPunks.addWhitelist(alice.address);
    await myPunks.connect(alice).mint();
  });

  it("should not be able to trnasfer myPunks",async function(){
    const zoreAdress = "0x0000000000000000000000000000000000000000";
    await expect(
      myPunks.connect(alice).transferFrom(alice.address,zoreAdress,1)
      .to.be.revertedWith("Transfer not allowed!")
    )
  });
})
```

测试

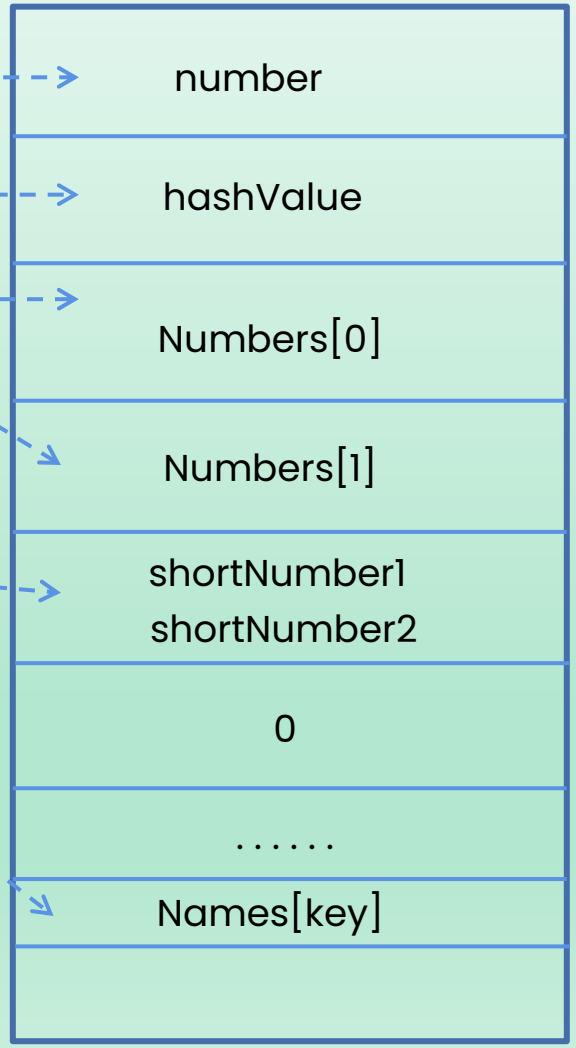
测试burn：

```
describe("Burn", function () {
  beforeEach(async function () {
    await mypunks.addWhitelist(alice.address);
    await mypunks.connect(alice).mint();
  });
  it("should be able to burn nft by owner", async function () {
    await expect(mypunks.burn(alice.address))
      .to.emit(mypunks, "PunkBurned")
      .withArgs(alice.address, 1);
  });
  it("should not be able to burn nft by non-owner", async function () {
    await expect(mypunks.connect(bob).burn(alice.address)).to.be.revertedWith(
      "Not owner"
    );
  });
});
```

04 代理与可升级合约

跨合约调用

```
1  uint256 public number;
2
3  bytes32 public hashValue;
4
5  uint256[2] public numbers = [1, 2];
6
7
8
9  uint8 public shortNumber;
10 uint8 public shortNumber2;
11
12 mapping(uint256 => string) public names;
```

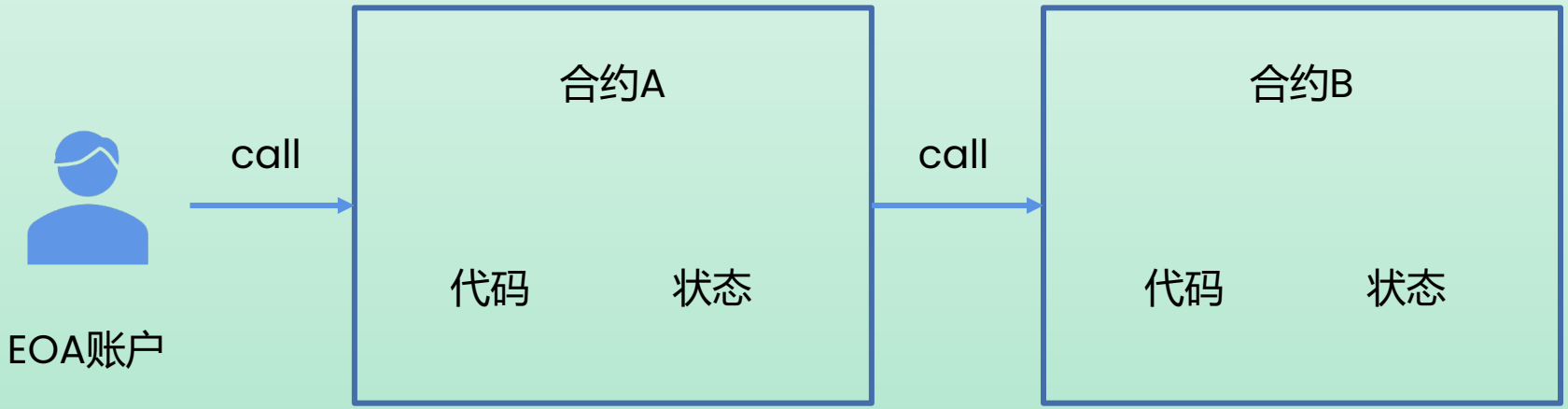


location = keccack256(key.slot)

跨合约调用

一笔交易流程：

- 交易**只能由EOA**账户发起
- 调用 = 执行代码 + 修改状态



核心问题: 执行谁的代码 & 修改谁的状态

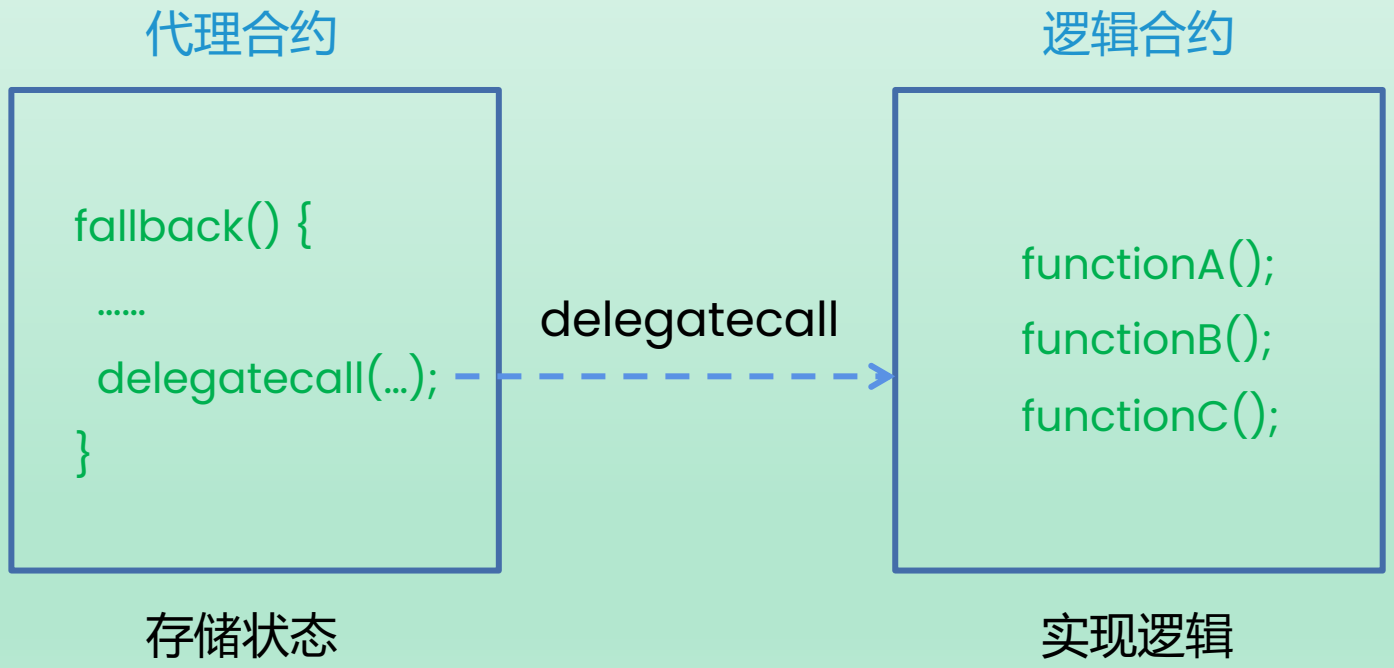
跨合约调用

三种跨合约调用方式：

调用方式	合约B中的msg.sender	合约B中的上下文	合约B的状态能否被更改	常用场景
Call	合约A	合约B	可更改	常规调用
Delegate Call	合约A	合约A	可更改	调用链接的Library
Static Call	合约A	合约B	不可更改	调用view或pure函数

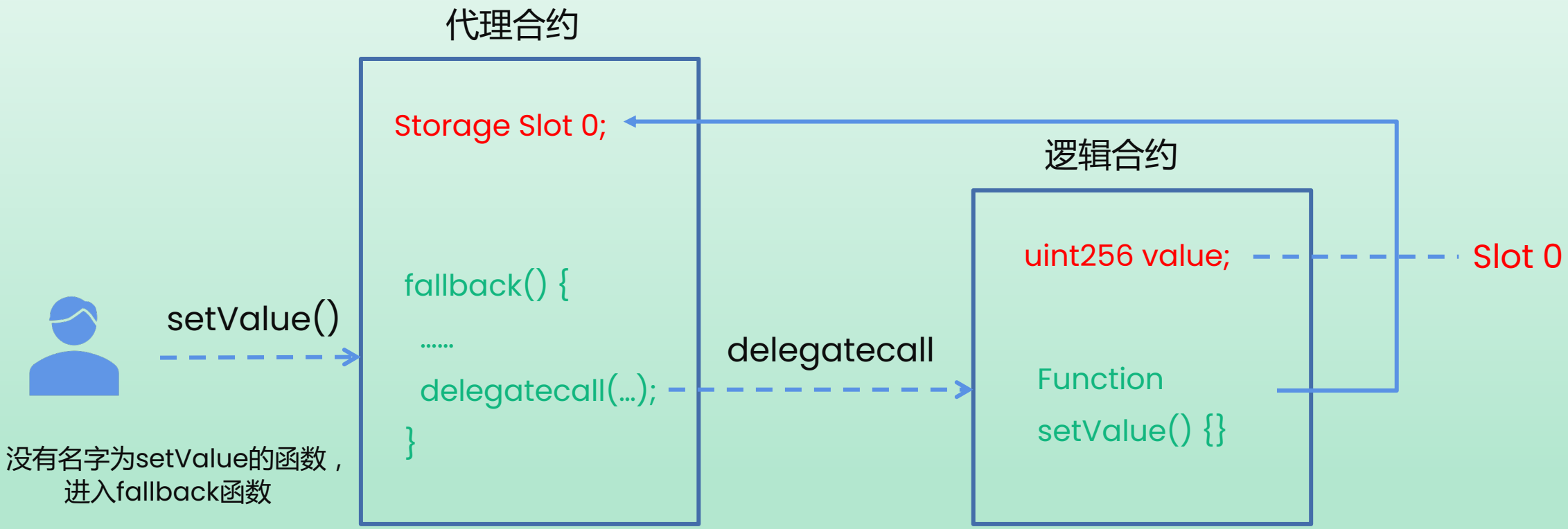
普通代理框架

- 使用fallback函数来转移调用的目标函数，避免重复定义接口
- 使用delegatecall来"借用"其他合约的逻辑，处理自己合约内部的状态



普通代理框架

实际改变的是代理合约的第一个存储插槽中的值，逻辑合约中的value并未改变



存储插槽冲突

当我们调用setNumber()函数时，会出现什么问题？

```solidity
address public logicContract;

constructor(address _logicContract) {
    logicContract = _logicContract;
}

function setLogicContract(address _logicContract) public {
    logicContract = _logicContract;
}

fallback() external {
    address _impl = logicContract;
    assembly {
        ...
    }
}
```

```solidity
uint256 public number;
string public name;

function setNumber(uint256 _number) public {
    number = _number;
}

function setName(string memory _name) public {
    name = _name;
}
```

！存储空间只有位置，没有名字，所以调用setNumber()会把logicContract变量修改掉！

存储插槽冲突

解决方案

把代理合约模式中所需要的特定的状态变量存储在靠后一个确定的插槽中

常见的两个地址

- 逻辑合约插槽

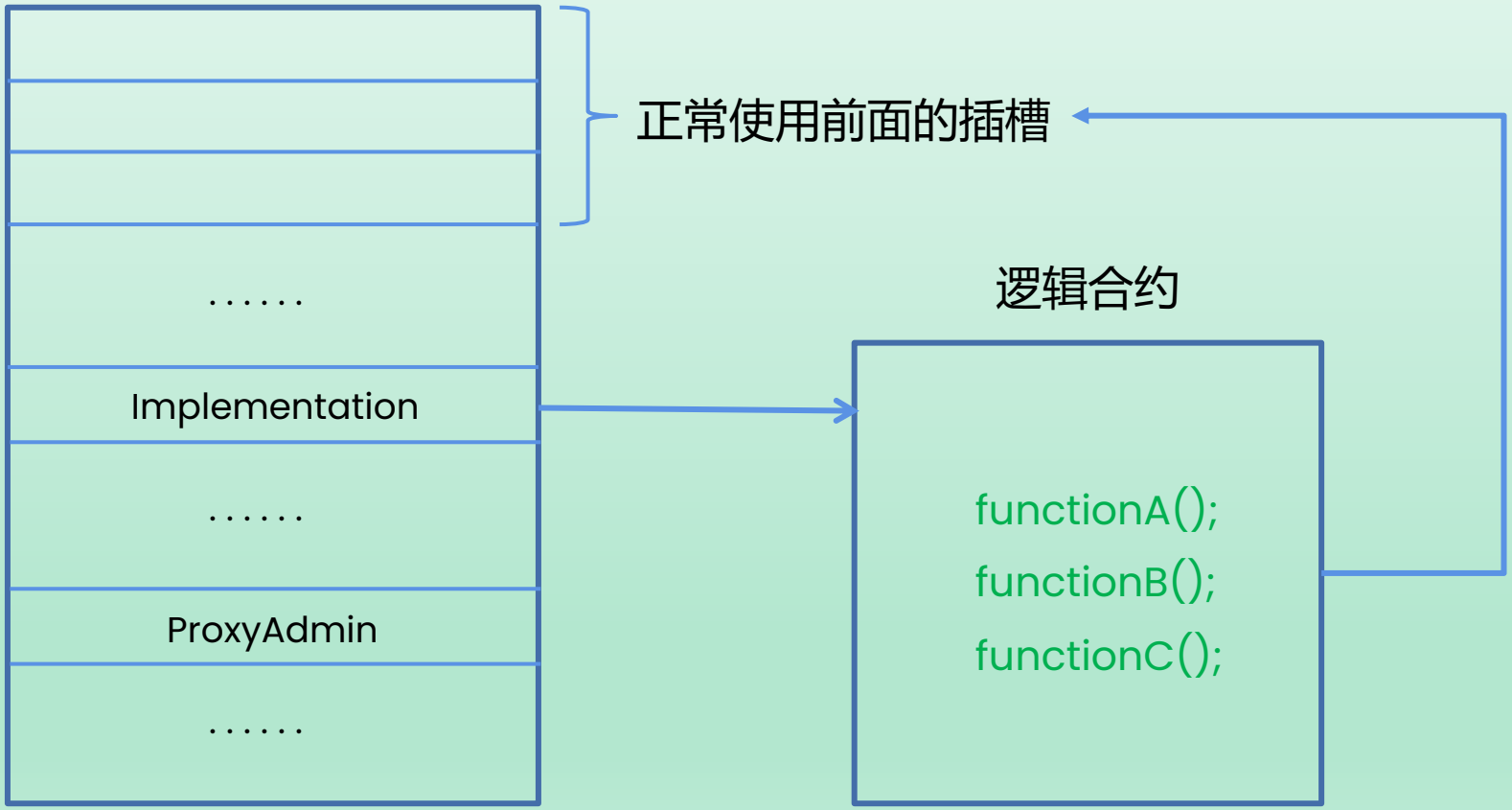
Implementation:
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc
= bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1))

- 代理管理合约插槽

ProxyAdmin:
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103
= bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1))

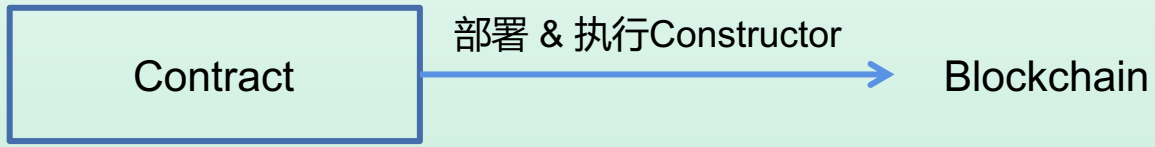
存储插槽冲突

EIP-1967标准下的存储插槽结构，插槽重合概率极小

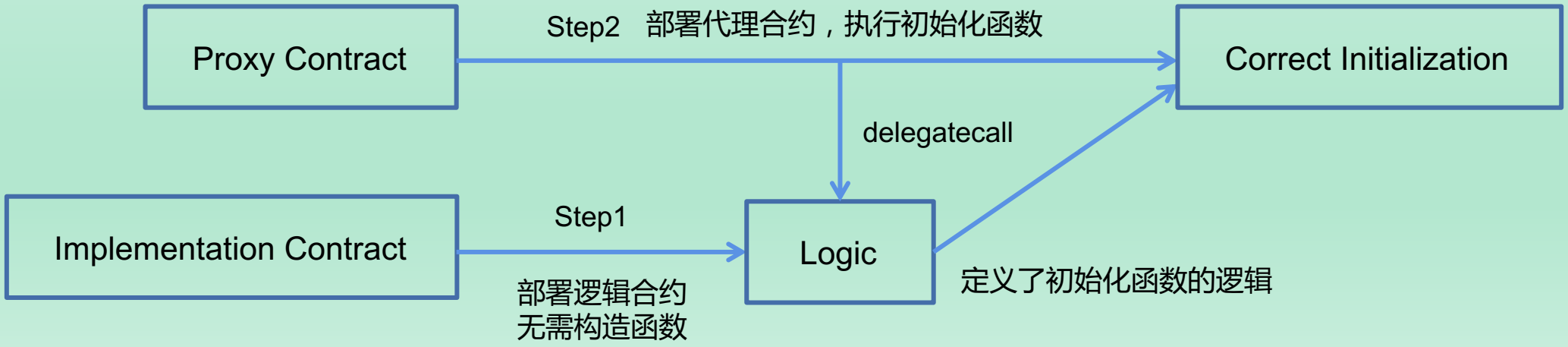


构造函数

构造函数: 在合约被部署时被调用，且仅有这一次调用



代理模式中，需要把构造函数变成初始化函数，并确保只初始化一次



构造函数

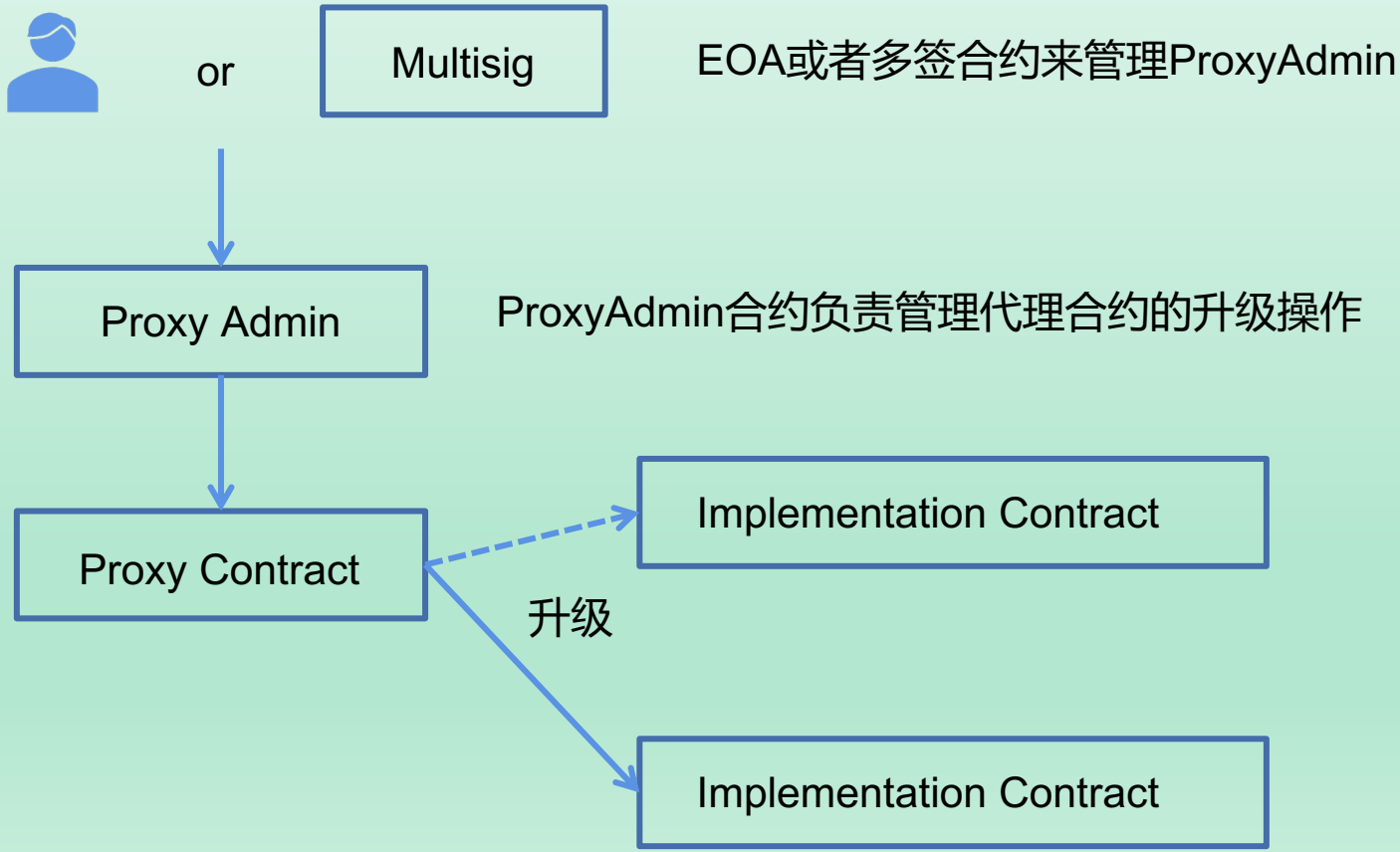
初始化函数示例

```
1  bool public initialized;
2
3  uint256 public value;
4
5  modifier initializer() {
6      require(!initialized, "Only initialize once");
7      _;
8      initialized = true;
9  }
10
11 function initialize(uint256 _initValue) public initializer {
12     value = _initValue;
13 }
14
```

初始化完成后，更改状态完成锁定

最佳实践

TransparentUpgradeableProxy 透明可升级代理模式



注意事项

可以做

- 添加新的函数、事件、错误等
- 更改已有函数的逻辑
- 在当前定义的最后一个状态变量后面添加新的状态变量

不可以做

- 移除现有的状态变量
- 更改现有的状态变量的类型、位置
- 更改继承的顺序
- 更改所继承的合约中的状态变量

05 ERC721可升级合约部署

准备

- 安装@openzeppelin/contracts-upgradeable

  $ npm install @openzeppelin/contracts-upgradeable

- 引入并继承Initializable, ERC721Upgradeable合约

```
1  pragma solidity ^0.8.9;
2
3  import "@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol";
4  import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5
6  contract MyPunksUpgradeable is Initializable, ERC721Upgradeable{
```

准备

- 将ERC721中的constructor替换为 initialize函数并调用 ERC721Upgradeable中的__ERC721_init 初始化函数，以便代理合约能够正常完成初始化。
- 添加构造器调用_disableInitializers() 作为额外保护， 防止实现合约本身被初始化。

```
// Locks the contract, preventing any future reinitialization.
// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();

}

function initialize() initializer public {
    __ERC721_init("MyPunks", "PUNK");
    owner =  msg.sender;
}
```

部署&升级

Hardhat中集成了可升级合约的组件我们可以

- upgrades.deployProxy直接部署我们的代理和实现合约
- upgrades.upgradeProxy切换实现合约

```
async function main() {
  const MyPunks = await ethers.getContractFactory("MyPunks");

  // Deploying    部署 V1
  const myPunks = await upgrades.deployProxy(MyPunks);
  await myPunks.deployed();

  console.log(
    `Deployed MyPunks to :${myPunks.address} with ${myPunks.deployTransaction.hash}`
  );
  // Upgrading    升级 V2
  const MyPunksV2 = await ethers.getContractFactory("MyPunksV2");
  const upgraded = await upgrades.upgradeProxy(myPunks.address, MyPunksV2);

  console.log(
    `Deployed MyPunks to :${upgraded.address} with ${upgraded.deployTransaction.hash}`
  );

}
```


BLOCKCHAIN ACADEMY
BEOSIN
Blockchain Security
课程练习

练习

- 使用Hardhat部署一个可升级的ERC721合约项目
- 用Hardhat的script或task完成白名单、铸造和转移操作
- 完成一次合约的升级，在升级后对新的代码逻辑完成新的铸造和转移操作


BLOCKCHAIN ACADEMY
BEOSIN
Blockchain Security
Thanks
talentre.academy
@Blockchain_AC
@Yang1127LI
Blockchain Academy
blockchain-academy-group