Solidity智能合约开发

第14.1课：众筹平台-实战开发

第三阶段　　实战项目　　Hardhat 3 完整开发流程

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.24;
contract CrowdfundingCampaign {
    enum State { Preparing, Active, Success, Failed, Closed }

    State public state;
    address public immutable owner;
function contribute() external payable {
    require(msg.value > 0, "Contribution must be positive");
```

讲师：区块链导师Layer

课程目标 Learning Objectives

需求分析和架构设计
掌握众筹平台的需求分析和架构设计能力

工厂合约开发
开发工厂合约创建多个众筹项目

前端界面开发
开发前端界面与合约交互，提升用户体验

状态机模式实现
实现完整的状态机模式，掌握状态转换逻辑

完整测试用例
编写完整的测试用例，确保合约安全性

测试网部署验证
完成测试网部署和验证，确保项目可上线

项目概述 Project Overview

核心功能

创建众筹项目
设置目标金额、截止时间和项目描述

用户参与众筹
贡献资金、记录贡献记录、实时更新进度

状态自动管理
自动处理募集中、成功、失败状态转换

资金安全处理
成功提取资金、失败自动退款机制

退款机制
项目失败时安全退还用户贡献资金

技术栈

Solidity 0.8.24
智能合约开发语言

Hardhat 3
以太坊开发框架

TypeScript
JavaScript超集，提供静态类型检查

React / Next.js
前端框架，构建用户界面

Ethers.js v6
与区块链交互的JavaScript库

需求分析 – 功能需求

项目创建

- 设置目标金额
- 设置截止时间
- 设置项目描述

参与众筹

- 用户贡献资金
- 记录贡献记录
- 实时更新进度

状态管理

- 募集中
- 成功
- 失败

资金处理

- 成功提取资金
- 失败自动退款
- 安全转账

查询功能

- 查询项目信息
- 查询贡献记录
- 查询状态

权限控制

- 项目创建者权限
- 参与者权限
- 状态转换权限

非功能需求 Non-Functional Requirements

安全性

- 防止重入攻击
- 防止整数溢出
- 权限验证

性能

- Gas优化
- 批量操作支持

可扩展性

- 工厂模式支持多项目
- 模块化设计

可测试性

- 完整测试覆盖
- 模拟环境支持

系统架构 System Architecture

前端层
(Frontend)

React / Next.js
用户界面

Ethers.js v6
区块链交互

→ 用户交互

合约层
(Smart Contracts)

CrowdfundingFactory
工厂合约

CrowdfundingCampaign
众筹合约

⇄ 状态转换

区块链层
(Blockchain)

Ethereum / Sepolia
区块链网络

交易处理
资金安全

🔒 不可篡改

合约架构设计

Contract Architecture

CrowdfundingFactory

```
contract CrowdfundingFactory {
    CrowdfundingCampaign[] public campaigns;
    mapping(address => uint256[]) public userCampaigns;

    function createCampaign(uint256, uint256) external
    returns (address);
}
```

creates

CrowdfundingCampaign

```
contract CrowdfundingCampaign {
    enum State { Preparing, Active, Success, Failed,
    Closed }
    State public state;
    address public immutable owner;
    function contribute() external payable;
function withdraw() external;
```

状态机设计

State Machine Design

工厂合约实现

CrowdfundingFactory

```
contract CrowdfundingFactory {
 CrowdfundingCampaign[] public campaigns;
 mapping(address => uint256[]) public userCampaigns;
```

核心函数

```
function createCampaign(uint256 _goal, uint256 _durationInDays)
external returns (address)
function getCampaigns() external view returns (address[] memory)

function getUserCampaigns(address user) externalview returns
(address[] memory)
```

数据结构

- campaigns[] – 存储所有众筹项目
- userCampaigns – 映射用户与其创建的项目

查询功能

- getCampaigns() – 获取所有项目
- getUserCampaigns() – 获取用户项目

完整测试 Comprehensive Testing

测试结构

单元测试

- 合约部署测试
- 状态转换测试
- 函数功能测试

集成测试

- 工厂合约创建项目
- 多用户交互测试
- 状态机完整流程

边界和Gas测试

- 零值处理、溢出保护
- 权限验证、Gas消耗分析

测试示例

```
// 测试部署和基础功能
describe("CrowdfundingCampaign", function () {
let campaign;
let owner;
beforeEach(async function () {
[owner] = await ethers.getSigners();
const CampaignFactory = await ethers.getContractFactory(

"CrowdfundingCampaign"
);
campaign = await CampaignFactory.deploy(
owner.address,
ethers.parseEther("10"),
7 // 7天
);
});
it("Should deploy with correct initial state", async function () {

expect(await campaign.state()).to.equal(0); // Preparing

expect(await campaign.owner()).to equal(owner.address);

});
```

部署和验证 Deployment & Verification

部署前检查

- 所有测试通过
- Gas优化完成
- 代码审查通过
- 环境变量配置
- RPC端点可用
- 测试网ETH充足

Hardhat Ignition部署

```
// ignition/modules/Crowdfunding.ts
import { buildModule } from "@nomicfoundation/hardhat-ignition/modules";

export default buildModule("CrowdfundingModule", (m) => {
  const factory = m.contract("CrowdfundingFactory");

  return { factory };
});
```

部署命令：

```
# 部署到Sepolia测试网
npx hardhat ignition deploy ignition/modules/Crowdfunding.ts --network sepolia --reset
```

合约验证

自动验证（部署时）：

```
npx hardhat ignition deploy ... --verify
```

手动验证：

```
npx hardhat verify --network sepolia <CONTRACT_ADDRESS>
<CONSTRUCTOR_ARGS>
```

```
etherscan: {
  apiKey: process.env.ETHERSCAN_API_KEY,
},
```

部署后测试

- 合约地址正确
- 所有函数可调用
- 状态查询正常
- 事件监听正常
- 前端连接正常
- 交易执行成功