Foundry框架学习

Solidity智能合约开发系列课程 – 第12.1课

第三阶段 | 开发框架与实战 | Foundry

讲师：[Layer]

本节课你将学到

- 理解 **Foundry** 与Hardhat的核心差异
- 掌握 **Foundry** 的安装和配置方法
- 熟悉Forge、Cast、Anvil三大工具
- 学会使用 **Foundry** 进行项目开发
- 掌握Solidity原生测试编写
- 学会使用Anvil本地测试网
- 熟练使用Cast命令行工具

为什么选择Foundry?

特性	Hardhat	Foundry
实现语言	JavaScript/TypeScript	Rust
编译速度	较慢	极快（10-100倍）
测试语言	JavaScript/TypeScript	Solidity原生
模糊测试	需插件	内置支持
本地节点	Hardhat Network	Anvil
部署脚本	JS/TS脚本	Solidity脚本
学习曲线	平缓	中等

Foundry的核心优势:

- **极致的性能**： Rust实现，编译速度极快
- **Solidity原生测试**： 直接用Solidity写测试
- **强大的工具链**： Forge、Cast、Anvil一体化
- **适合场景**： 纯合约开发、需要极致性能

安装Foundry

安装步骤

1 **步骤1：下载安装脚本**

```
curl –L https://foundry.paradigm.xyz | bash
```

2 **步骤2：运行foundryup**

```
foundryup
```

3 **步骤3：验证安装**

```
forge --version
cast --version
anvil --version
```

系统要求

- Rust（安装脚本会自动安装）
- 建议Node.js >= 18（用于前端工具协作）
- 足够的磁盘空间（约500MB）

常见问题

- **问题：curl命令失败**
  - 解决：检查网络连接，或使用代理
- **问题：权限错误**
  - 解决：使用sudo或检查PATH配置
- **问题：Rust安装失败**
  - 解决：手动安装Rust：curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

Forge：核心开发工具

Forge是什么？

Forge是Foundry的核心工具，提供：

- 合约编译
- 测试执行
- 脚本运行
- 部署管理

核心功能

1. 编译系统

- 支持多版本Solidity
- 增量编译
- 自动依赖管理

2. 测试框架

- Solidity原生测试
- 模糊测试（Fuzzing）
- 属性测试

3. 脚本系统

- Solidity脚本（.s.sol）
- 支持发送真实交易
- 模拟执行（dry-run）

主要命令

```
# 初始化项目
forge init
```

```
# 编译合约
forge build
```

```
# 运行测试
forge test
```

```
# 运行脚本
forge script
```

```
# 清理构建
forge clean
```

性能对比（与Hardhat）

特性	编译速度	测试速度	内存占用
Forge	10-50倍	10-100倍	更低
Hardhat	基准	基准	基准

Cast & Anvil：强大的辅助工具

Cast：命令行工具

核心功能

- 与区块链交互
- 编码/解码数据
- 签名和验证
- Gas估算

常用命令示例

```
cast chain-id --rpc-url $RPC_URL
```

```
cast balance 0x... --rpc-url $RPC_URL
```

Anvil：本地测试节点

核心特性

- 高性能本地以太坊节点
- 10个预置测试账户
- 支持主网分叉
- 可配置区块时间

常用命令

```
anvil
```

```
anvil --port 8546 --chain-id 31337
```

Anvil vs Hardhat Network

特性	Hardhat Network	Anvil
性能	一般	更快
分叉	基础	更灵活

创建Foundry项目

项目初始化步骤

步骤1： 创建项目目录mkdir foundry-demo

```
cd foundry-demo
```

步骤2： 初始化项目forge init

```

```

初始化选项

forge init --no-commit 不创建Git仓库

forge init --no-template 不包含示例代码

forge init --no-commit --no-template 同时使用两个选项

项目结构

```
foundry-demo/
    src/ 合约目录
    test/ 测试目录
    script/ 脚本目录
    lib/ 依赖目录
    foundry.toml 配置文件
    .gitignore Git忽略文件
    README.md 项目说明
```

目录说明

src/ 智能合约目录
存放所有.sol合约文件，支持子目录组织

test/ 测试文件目录
测试文件命名：ContractName.t.sol

script/ 部署脚本目录
脚本文件命名：ScriptName.s.sol

lib/ 依赖库目录
使用forge install安装依赖

foundry.toml配置文件

基础配置

```
[profile.default]
src = "src"
out = "out"
solc = "0.8.24"
optimizer = true
optimizer_runs = 200
```

多Profile配置

```
[profile.ci]
optimizer = true
optimizer_runs = 1
[profile.production]
optimizer = true
optimizer_runs = 10000
[profile.dev]
optimizer = false
```

网络配置

```
[rpc_endpoints]
sepolia = "${SEPOLIA_RPC}"
mainnet = "${MAINNET_RPC}"
localhost = "http://127.0.0.1:8545"
[etherscan]
sepolia = { key = "${ETHERSCAN_API_KEY}" }
```

编译器配置

```
[profile.default]
solc_version = "0.8.24"
solc = [ "0.8.24" , "0.7.6" ]
auto_detect_solc = true
```

配置开发网络

环境变量设置

创建 .env 文件：

```
SEPOLIA_RPC=https://sepolia.infura.io/v3/YOUR_PROJECT_ID

MAINNET_RPC=https://mainnet.infura.io/v3/YOUR_PROJECT_ID

PRIVATE_KEY=your-private-key-here

ETHERSCAN_API_KEY=your-etherscan-key
```

在 foundry.toml 中使用：

```
[rpc_endpoints]

    sepolia = "${SEPOLIA_RPC}"

    mainnet = "${MAINNET_RPC}"

    [etherscan]

    sepolia = { key = "${ETHERSCAN_API_KEY}" }
```

网络类型

Anvil本地网络：
URL: http://127.0.0.1:8545
Chain ID: 31337

分叉主网：
anvil --fork-url $MAINNET_RPC

主网配置：
[rpc_endpoints]
mainnet = "${MAINNET_RPC}"

安全提示

私钥安全：
使用环境变量存储私钥
不要提交 .env 到 Git

RPC端点：
使用可靠的RPC提供商
Infura、Alchemy、QuickNode

使用网络：

```
forge script script/Deploy.s.sol --rpc-url $SEPOLIA_RPC
```

编译智能合约

基础编译

```
// 编译所有合约
forge build

// 清理后重新编译
forge clean && forge build

// 查看编译详情
forge build --sizes
```

编译输出示例

```
Compiling 2 files with 0.8.24
Compiler run successful
Artifacts written to /path/to/out
Size of Counter.sol:Counter: 0.2 KB
```

编译选项

- 查看字节码大小： forge build --sizes
- 查看合约列表： forge build --names
- 强制重新编译： forge build --force
- 编译特定合约： forge build src/Counter.sol

编译优化

在 中添加配置选项:

optimizer = true

optimizer_runs = 200

via_ir = true

常见编译错误

- Contract constructor arguments too long : 缩短参数
- Stack too deep : 减少局部变量
- Compiler run failed : 检查文件路径

编译产物

- artifact 文件（ABI、字节码）
- size 文件（合约大小分析）
- metadata 文件（部署元数据）
- compact-format 文件（紧凑格式输出）

Solidity部署脚本

基础部署脚本

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.24;

import "forge-std/Script.sol";
import "../src/Counter.sol";

contract DeployCounter is Script {
    function run() external {
        uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");
        vm.startBroadcast(deployerPrivateKey);

        Counter counter = new Counter();

        vm.stopBroadcast();
        console.log("Counter deployed at:", address(counter));
    }
}
```

脚本执行

模拟执行 (Dry-run)

```
forge script script/DeployCounter.s.sol
forge script script/DeployCounter.s.sol --fork-url $SEPOLIA_RPC
```

发送真实交易

```
forge script script/DeployCounter.s.sol --rpc-url $SEPOLIA_RPC --broadcast
forge script script/DeployCounter.s.sol --rpc-url $SEPOLIA_RPC --broadcast --verify
```

多网络部署

```
forge script script/DeployCounter.s.sol --rpc-url sepolia --broadcast
forge script script/DeployCounter.s.sol --rpc-url mainnet --broadcast
```

脚本位置

脚本位于 script/DeployCounter.s.sol

脚本中的Cheatcodes

环境变量

vm.envUint("PRIVATE_KEY")
vm.envAddress("OWNER")
vm.envString("NAME")

地址操作

vm.addr(privateKey)
vm.label(address, "Label")

区块操作

vm.roll(blockNumber)
vm.warp(timestamp)

账户操作

vm.deal(address, amount)
vm.prank(address)

Anvil本地节点

启动Anvil

```
anvil
```

默认配置

- HTTP Server:
- WebSockets Server:
- Private Keys: 10个预置账户
- Chain ID: 31337 (本地)

账户管理

- 查看预置账户:
- 设置自定义私钥:

自定义配置

- 指定区块时间:
- 设置初始余额:
- 指定端口:

分叉主网

- 分叉以太坊主网:
- 指定区块分叉:

状态管理

- 导出状态:
- 导入状态:

部署到测试网

准备工作

1. **获取测试ETH**（Sepolia水龙头）：

```
https://sepoliafaucet.com
https://faucet.quicknode.com/ethereum/sepolia
```

2. **配置环境变量**（.env文件）：

```
SEPOLIA_RPC=https://sepolia.infura.io/v3/YOUR_PROJECT_ID
PRIVATE_KEY=your-testnet-private-key
ETHERSCAN_API_KEY=your-etherscan-api-key
```

3. **配置foundry.toml**：

```
[rpc_endpoints]
sepolia = "${SEPOLIA_RPC}"

[etherscan]
sepolia = { key = "${ETHERSCAN_API_KEY}" }
```

部署流程

1. **模拟执行**（Dry-run）：

```
forge script script/DeployCounter.s.sol --rpc-url sepolia --account default
```

2. **检查Gas估算**：

```
forge script script/DeployCounter.s.sol --rpc-url sepolia --gas-limit 3000000
```

3. **发送交易**：

```
forge script script/DeployCounter.s.sol --rpc-url sepolia --broadcast --verify
```

部署后验证

```
查看部署的合约： cast code <contract-address> --rpc-url sepolia
```

```
调用合约函数： cast call <contract-address> "number()" --rpc-url sepolia
```

验证合约与链上交互

合约验证

自动验证（推荐）
在部署命令中添加--verify

```
forge script script/DeployCounter.s.sol --verify
```

手动验证
使用forge verify-contract命令

```
forge verify-contract <address> <contract-name>
--chain sepolia --etherscan-api-key $ETHERSCAN_API_KEY
```

带构造函数参数验证
使用十六进制参数或参数文件

```
forge verify-contract --constructor-args $(cast abi-encode
"constructor(uint256)" 100)
```

使用Cast链上交互

查询链上数据
余额、链ID、区块号、Gas价格

```
cast balance 0x... --rpc-url sepolia
```

调用只读函数
查询合约状态

```
cast call 0x... "number()"
```

发送交易
执行合约写操作

```
cast send 0x... "increment()"
--private-key $PK --rpc-url sepolia
```

Gas估算
预估操作Gas成本

```
cast estimate 0x... "increment()"
```

编码/解码
ABI编码解码

```
cast abi-encode "transfer(address,uint256)"
0x... 1000
```

签名和验证
消息签名与验证

```
cast wallet sign-message "Hello World"
```

Cast：强大的命令行工具

链上查询

查询链ID、区块号、Gas价格等区块链数据

```
cast chain-id --rpc-url $RPC_URL
```

合约交互

调用合约函数、发送交易、估算Gas

```
cast call 0x... "name()" --rpc-url $RPC_URL
```

数据编码

ABI编码、解码、函数选择器编码

```
cast abi-encode "transfer(address,uint256)" 0x... 1000
```

地址和私钥

私钥转地址、生成随机私钥、地址格式转换

```
cast wallet address --private-key $PK
```

数值转换

Wei与ETH互转、十六进制与十进制转换

```
cast --to-unit 1000000000000000000 ether
```

签名和验证

签名消息、验证签名、签名交易

```
cast wallet sign-message "Hello"
```

交易相关

发送ETH、发送原始交易、查询交易

```
cast send 0x... --value 1ether --private-key $PK
```

日志和事件

查询事件日志、解码日志

```
cast logs --address 0x... --event "Transfer(address,address,uint256)"
```

总结与最佳实践

核心要点回顾

- **Foundry** vs Hardhat：性能极快，Solidity原生测试
- 三大工具：Forge（编译测试）、Cast（命令行）、Anvil（本地节点）
- 开发流程：初始化 → 编写合约 → 编写测试 → 编译 → 部署 → 验证
- 配置管理：foundry.toml、环境变量、多Profile

最佳实践

- 使用环境变量管理私钥和API密钥
- 先dry-run再broadcast，避免意外
- 使用--slow避免nonce冲突
- 及时验证部署的合约
- 编写完整的测试用例，使用模糊测试

性能优化建议

- 编译优化：增量编译，合理设置optimizer_runs
- 测试优化：并行测试，合理设置fuzz runs
- 部署优化：批量部署，CREATE2确定性地址

学习资源

- 官方文档：详细API参考和最佳实践
- 社区资源：Foundry Discord、GitHub讨论
- 实战项目：Uniswap V3、Solmate

深度探索

- 模糊测试深入
- 属性测试与覆盖率分析
- Gas优化技巧