FISCO BCOS Documentation

发布 ***v3.1.0***

fisco-dev

2022 年 **12** 月 **19** 日

Contents

CHAPTER 1

Truora介绍

1.1 分支版本说明

！(首先确认区块链底层对应的版本，并切换到对应的文档和代码分支进行操作)！

面向区块链底层 FISCO-BCOS 2.6.0 +

- 区块链底层对应 FISCO-BCOS 2.6.0 +
- 文档连接v2stable
- Truora-Service 稳定代码分支v2stable
- 一键安装，Truora docker和web相关模块参见v2stable文档里的相关说明

面向区块链底层 FISCO-BCOS 3.1.x +

- 区块链底层对应 FISCO-BCOS 3.1及以上
- 文档连接Latest/v3dev分支
- Truora-Service 主分支main
- Truora-Service 开发分支v3dev
- v3dev分支暂不包括一键安装、docker和web模块，欢迎参与开发贡献

！(首先确认区块链底层对应的版本，并切换到对应的文档和代码分支进行操作)！

1.2 预言机简介

预言机简介

区块链是一个确定性的、封闭的系统环境，智能合约不管何时何地运行都必须是一致的结果，所以虚拟机（VM）不能让智能合约有网络调用，不然结果就是不确定的。

智能合约不能直接获取到链外真实世界的数据，导致区块链与现实世界是割裂的，也极大地限制了区块链的应用场景。 而如何将区块链和现实世界连接起来，就需要引入预言机服务，通过预言机将现实世界的数据输入到区块链上，为智能合约提供与外部世界的连接性。

中国人民银行发布的《区块链能做什么？不能做什么？》报告中，是这样对预言机定义的： “区块链外信息写入区块链内的机制，一般被称为预言机 (oracle mechanism)。”

1.3 Truora简介

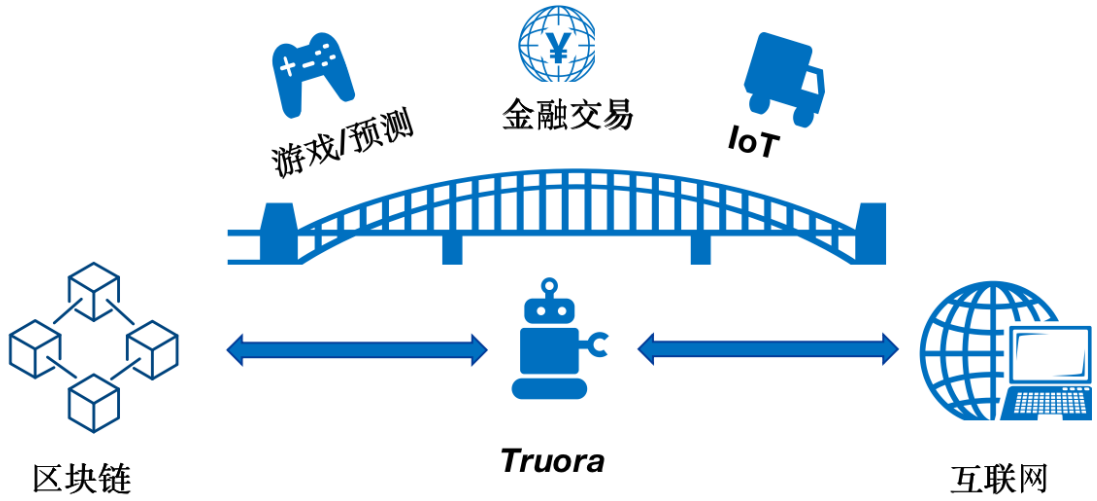
Truora简介

Truora 是 FISCO-BCOS 区块链平台的预言机服务解决方案，是在广泛调研的基础上针对 **联盟链** 场景设计的可信预言机服务。

区块链愈发展，对链下数据的需求就会愈强烈，预言机的重要性也会愈发凸显。

作为连接 *FISCO-BCOS* 联盟链和互联网的桥梁，Truora致力于解决互联网数据安全可信的上链，助力扩宽联盟链的应用场景和丰富联盟链的生态。

Truora 服务主要由后端 *Java* 组件 Truora-Service 和前端 *Vue* 组件 Truora-Web 组成。



1.4 设计原则

设计原则

Truora是一整套预言机方案解决的集合，包含中心化和去中心化部署，用户可以针对不同的业务场景选择适合的部署方式。 此外，不局限于解决互联网数据上链，会结合联盟链场景制定数据提供商的规范，如数据格式规范，治理规范。以给联盟链提供可信可验证的优质数据。

中心化部署 针对请求时延低，信任要求不是很高的场景，主要着力于搭建方便，结合联盟链场景，主要问题是解决中心化机构作恶问题。

- 软件上会深入研究TLS技术，并进行改造。 暴露连接细节以及链上数据验签。
- 硬件上支持TEE等安全环境下的部署。

去中心化部署 主要分为链上聚合和链下聚合

对于数据方面 支持获取互联网上的数据，同时也会制定联盟链数据提供标准，引入优质数据服务提供商，为联盟链提供优质可信的数据服务。

1.5 主要特性

主要特性

- 链下 HTTPS API 数据获取
- 支持多链多群组服务（必须同为 *ECDSA* 或 国密）
- 支持VRF随机数生成
- 支持国密
- 支持集群部署
- 支持中心化部署和去中心化部署
- 支持多数据格式访问
- 支持请求状态查询

1.6 应用场景

应用场景

- **游戏/预测市场方面：**获取链上安全的随机数，智能合约实现更公平游戏场景。
- **物联网方面：**对于IoT应用，将传感器信息上链，智能合约验证并触发下一步的行为。
- **供应链金融方面：**获取链下订单信息和汇率信息等。

1.7 路线图

技术路线

目前已完成中心化方式获取链下数据， Truora在2021年度的技术路线如下：

- **Q1–Truora中心化预言机功能**

  主要完成中心化预言机功能，支持国密和非国密环境下的链下API访问和VRF功能，支持监听多链多群组，支持请求的加密，预言机采集数据签名和链上验签。

- **Q2–Truora去中心化预言机功能**

  主要完成去中心化预言机的链上聚合功能。支持Truora的分布式部署以及结果的链上聚合（求平均，中位数，最大值，最小值），和数据提供商规范制定， 增加AMOP的支持。

- **Q3–Truora解决方案**

  主要完成基于Truora应用实例开发。如物联网，供应链金融，游戏，积分，NFT等场景结合预言机开发具体的应用实例。

- **Q4–Truora去中心化预言机功能**

  主要完成去中心化预言机的链下聚合功能，增加p2p网络以及密码学套件，支持BLS门限签名技术，实现链下聚合功能。

CHAPTER 2

安装部署

！(首先确认区块链底层对应的版本，并切换到对应的文档和代码分支进行操作)！

>> 面向区块链底层 FISCO-BCOS 2.6.0 +

- 区块链底层对应 FISCO-BCOS 2.6.0 +
- 文档连接v2stable
- Truora-Service 稳定代码分支v2stable
- 一键安装，Truora docker和web相关模块参见v2stable文档里的相关说明

>> 面向区块链底层 FISCO-BCOS 3.1.x +

- 区块链底层对应 FISCO-BCOS 3.1及以上
- 文档连接Latest/v3dev分支
- Truora-Service 主分支main
- Truora-Service 开发分支v3dev
- v3dev分支暂不包括一键安装、docker和web模块，欢迎参与开发贡献

！(首先确认区块链底层对应的版本，并切换到对应的文档和代码分支进行操作)！

2.1 操作步骤

Truora 服务的部署模式有如下两种方式：

- **一键部署**

适用场景

- 快速体验 Truora 服务
- 预言机服务的开发和调试

一键部署使用一键部署脚本（`deploy-all.sh`），在部署 Truora 服务时，会自动部署依赖服务，包括：

- **独立部署**

适用场景

- 已有 FISCO-BCOS 底层节点

独立部署使用脚本工具（`deploy-single.sh`），仅仅部署 Truora 的 **两个** 核心服务：

2.1.1 一键部署

重要:

- 使用一键部署 Truora 服务时，**仅支持 Linux 操作系统！！**
- 使用一键部署 Truora 服务时，**数据库服务无需配置和改动!!**。

部署介绍

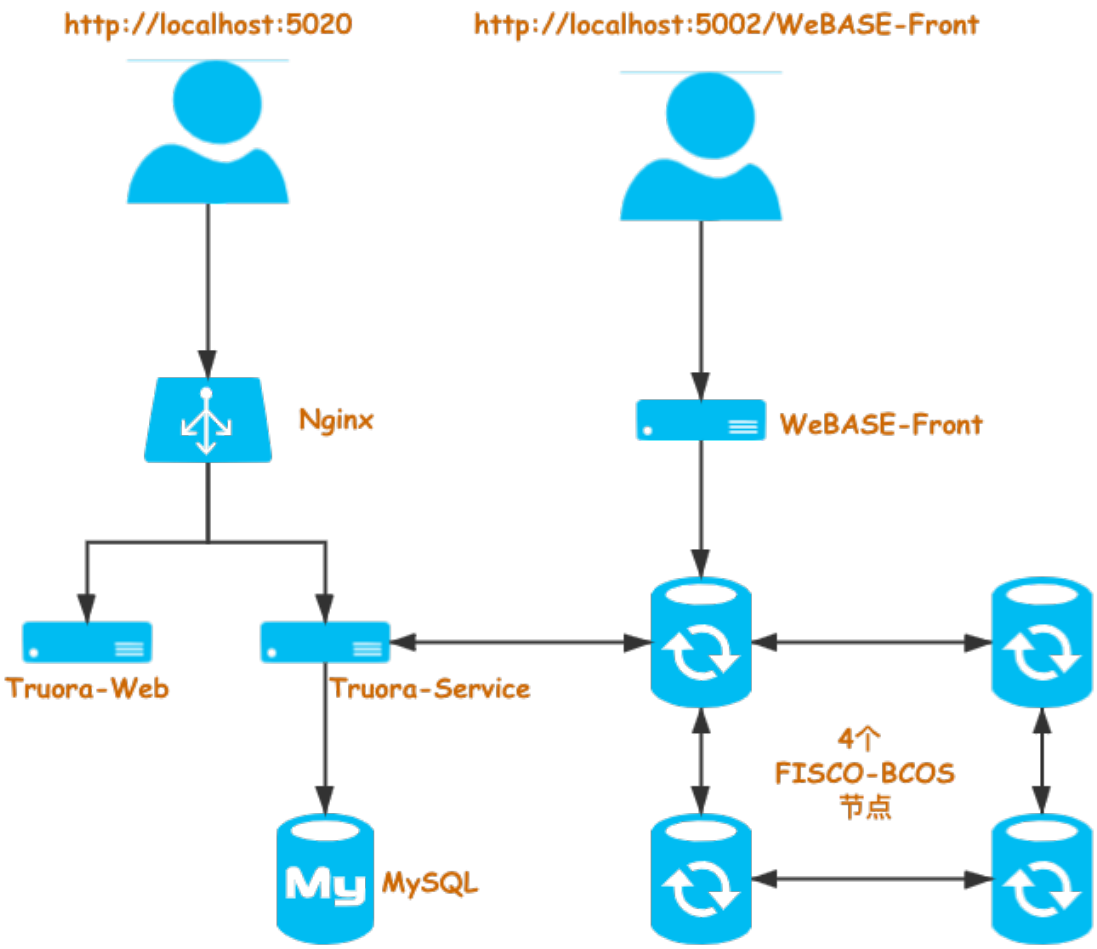
一键部署是基于 `Docker`，`Docker-Compose` 和 `Bash Shell` 封装的一个部署工具，提供一键部署整个 Truora 服务，包括依赖的 FISCO-BCOS 节点和 MySQL 数据库等。

适合以下场景：

- 快速体验 Truora 服务
- 预言机服务的开发和调试

使用一键部署工具，部署 Truora 服务时，会同时部署一个 WeBASE-Front（Solidity 合约的开发和调试环境）服务，作为 Truora 合约的开发和调试环境。

部署后的架构如下：



- **WeBASE-Front**

  WeBASE-Front 是 WeBASE 中间件的一个子系统服务，针对 FISCO-BCOS 区块链服务提供 Solidity 合约的可视化开发，编译，部署和调试功能。

  在进行 Truora 相关业务的合约开发和调试时，可以使用 WeBASE-Front 中的合约 IDE，方便合约的开发和调试，提高开发效率。

  关于 WeBASE-Front，请参考：WeBASE-Front

- **Truora-Web**

  Truora-Web 是 Truora 服务的前端 Web，主要包含以下几个功能：

  - 分页查询 预言机 请求历史记录
  - 单个查询 预言机 请求明细（状态，响应结果，错误信息等）
  - Truora-Service 内置合约地址查询
  - 查询所有 Truora-Service 服务列表

  关于 Truora-Web，请参考：*Truora-Web*

- **Truora-Service**

  Truora-Service 是 Truora 的服务端：

  - 监听 FISCO-BCOS 链上 Truora 合约的事件
  - 接收链上事件，调用 Http API 接口或 VRF 随机数生成库，获取结果

- 结果上链，供用户合约查询

关于 Truora 原理，请参考：*Truora-Service*

前置要求

系统要求

硬件配置

脚本说明

Truora 一键部署工具特性：

- 提供自动安装依赖服务功能，包括：`OpenSSL`, `curl`, `wget`, `Docker`, `Docker Compose` 等
- 调用 FISCO-BCOS 一键部署脚本 `build_chain.sh`，部署 4 个区块链底层节点
- 部署 WeBASE-Front 服务
- 部署 Truora-Service, Truora-Web 服务
- 部署 MySQL 服务
- 支持国密选项

关于脚本详细参数列表，请参考：脚本参数

获取部署脚本

部署脚本的获取方式包括：

- 下载部署包（**推荐使用**）
- Github 仓库拉取源码

```
## 从 GitHub 下载最新部署包
wget "https://github.com/WeBankBlockchain/Truora-Service/releases/download/v1.1.0/
↪docker-deploy.zip"

## 解压部署包
unzip docker-deploy.zip
```

如果需要下载指定版本，在 版本列表中 选择相应版本下载。

提示

- 由于网络原因，如果遇到打不开 Github 页面，或者无法从 GitHub 下载，可以从 CDN 下载。关于 CDN 说明，请参考：CDN – 部署工具包

如果需要从代码仓库，拉取部署包源码，请参考：部署工具源码

部署

进入部署脚本（`deploy_all.sh`）所在目录，执行命令：

```
# 自动安装依赖服务，默认从 CDN 拉取 Docker 镜像
# Docker Hub 官方仓库拉取镜像时，不仅速度比较慢，同时成功率也相对较低
#
# -d : 自动安装系统依赖
# -g : 使用国密
# -k : 从 Docker Hub 官方仓库拉取 Docker 镜像
bash deploy_all.sh -d
```

关于脚本详细参数列表，请参考：脚本参数

提示

- 重复执行部署脚本时，会提示某些目录已经存在，请根据提示输入字母：b [backup] 或者 d [Delete] 进行操作。

服务启停

如果一键部署脚本 `deploy_all.sh` 执行成功后显示 `Deploy Truora service SUCCESS!!`，表示部署成功。

- 使用 `bash start.sh` 启动 Truora 服务。
- 使用 `bash stop.sh` 停止服务。

在启动时，脚本会依次启动服务，并检测服务启动结果。

如果提示 `Truora service start up SUCCESS !!`，表示 Truora 服务启动成功。

如果启动失败，根据命令行的提示，检查启动失败服务的日志。关于查看服务的日志，请参考：日志查看

访问服务

Truora 服务启动成功后，打开浏览器，输入 `http://[IP]:5020`，比如：`http://127.0.0.1:5020`，访问 Truora-Web 服务，请参考：Truora-Web 服务介绍

提示

- 注意替换服务器的 *IP* 地址

关于 Truora 服务的 **开发教程**，请参考：Truora 开发教程

2.1.2 独立部署

提示

- 由于 Docker 的网络限制，使用独立部署 Truora 服务时，**仅支持 Linux 操作系统！！**

重要:

- 使用独立部署功能的用户，需要对 MySQL 和 FISCO-BCOS sdk 证书有一定了解。

安装介绍

独立部署是基于 Docker，Docker-Compose 和 Bash Shell 封装的一个部署工具，提供一键部署 Truora 服务，并连接到已有 FISCO-BCOS 链。

适合以下场景：

- 已有 FISCO-BCOS 底层节点

使用独立部署工具，部署 Truora 服务时，会部署 Truora-Service 和 Truora-Web 服务，此外，可以选择是否部署一个 MySQL 服务。

重要:

- 使用独立部署时，Truora-Service 需要链接到 FISCO-BCOS 节点。需要手动提供链接节点的 sdk 相关文件。
- 使用独立部署时，如果选择 **不部署** MySQL，在部署时，需要提供 MySQL 的链接信息，包括：IP，端口，用户名和密码。

部署的服务包括：

- Truora-Web
- Truora-Service
- MySQL（**可选**）

前置要求

系统要求

硬件配置

脚本说明

Truora 独立部署工具特性：

- 提供自动安装依赖服务功能，包括：OpenSSL, curl, wget, Docker, Docker Compose 等
- 部署 Truora-Service, Truora-Web 服务
- 部署 MySQL 服务（可选）
- 支持国密选项

关于脚本详细参数列表，请参考：脚本参数

获取部署脚本

部署脚本的获取方式包括：

- 下载部署包（**推荐使用**）
- Github 仓库拉取源码

```
## 从 GitHub 下载最新部署包
wget "https://github.com/WeBankBlockchain/Truora-Service/releases/download/v1.1.0/
↪docker-deploy.zip"

## 解压部署包
unzip docker-deploy.zip
```

如果需要下载指定版本，在 版本列表中 选择相应版本下载。

提示

- 由于网络原因，如果遇到打不开 Github 页面，或者无法从 GitHub 下载，可以从 CDN 下载。关于 CDN 说明，请参考：CDN – 部署工具包

如果需要从代码仓库，拉取部署包源码，请参考：部署工具源码

部署

执行部署脚本

- 进入部署脚本（deploy_single.sh）所在目录，执行命令：

```
# 自动安装依赖服务，默认从 CDN 拉取 Docker 镜像
# Docker Hub 官方仓库拉取镜像时，不仅速度比较慢，同时成功率也相对较低
#
# -d : 自动安装系统依赖
# -g : 使用国密
# -m : 部署 MySQL 服务
# -k : 从 Docker Hub 官方仓库拉取 Docker 镜像
bash deploy_single.sh -d
```

关于脚本详细参数列表，请参考：脚本参数

重要:

- 如果使用 *-g* 参数部署 Truora 国密版本时，需要确保 **链类型：国密**，**连接类型：国密**。关于链类型，国密类型，请参考：加密类型

配置证书目录

重要:

- 部署脚本 **不会检查 SDK 文件和需要连接的链是否匹配**，只会检查必须的证书文件是否存在。

非国密 Truora（ECDSA）

提示输入 SDK 目录（目录需要存在对应链的 ca.crt，node.crt，node.key 文件），输入目录后，按 **回车** 确认：

```
.......
================================================================================================
[INFO] Deploy services ...
[INFO] Enter certifications info.

[INFO] Enter sdk path:

# 提示输入 SDK 目录，回车确认
e.g:[ /root/webank/deploy/deploy/fiscobcos/nodes/127.0.0.1/sdk ]:

.......
```

脚本会自动检测输入的 SDK 目录中是否存在 ca.crt，node.crt，node.key 文件。

国密 Truora（SM2）

提示输入 gm SDK 目录（目录需要存在对应链的 gmca.crt，gmensdk.crt，gmensdk.key，gmsdk.crt，gmsdk.key 文件），输入目录后，按 **回车** 确认：

```
================================================================================================
[INFO] Deploy services ...
[INFO] Enter certifications info.

[INFO] Enter sdk path:

# 提示输入 gm SDK 目录，回车确认
e.g:[ /root/webank/deploy/deploy/fiscobcos/nodes/127.0.0.1/sdk/gm ]:

.......
```

脚本会自动检测输入的 gm SDK 目录中，是否存在 gmca.crt，gmensdk.crt，gmensdk.key，gmsdk.crt，gmsdk.key 文件。

配置 MySQL

如果未使用 -m 参数，会提示输入 MySQL 的连接信息。直接 **回车** 使用默认值：

```
.......
[INFO] User external MySQL.

Enter MySQL IP, default: 127.0.0.1 ? 127.0.0.1

Enter MySQL port, default: 3306 ?

Enter MySQL user, default: truora ?

Enter MySQL password, default: defaultPassword ? user

.......
```

在 MySQL 中创建一个 truora 数据库，供 Truora-Service 服务使用。

```
# 连接 MySQL
mysql -u root -p

# 创建 truora 数据库
CREATE DATABASE `truora` /*!40100 DEFAULT CHARACTER SET utf8mb4 */;
```

配置节点连接

修改 FISCO-BCOS 节点连接信息，编辑 truora/docker-compose.yml 文件：

```
.....
      # FISCO-BCOS 节点 IP，默认：127.0.0.1
      - "FISCO_BCOS_IP=127.0.0.1"
      # FISCO-BCOS 节点端口，默认：20200
      - "FISCO_BCOS_PORT=20200"
      # FISCO-BCOS 连接群组，默认：1
      - "FISCO_BCOS_GROUP=1"
.....
```

多链配置

使用独立部署的 Truora 服务，支持同时连接多链，以及一条链的多个群组。

挂载证书

在添加多链时，需要先将新链的证书目录挂载到 Docker 容器中。

在部署时生成的挂载文件，根据加密方式不同，生成的 Docker Compose 文件不同：

```
# 非国密 Truora（ECDSA）
truora/deploy/docker-compose-ecdsa.yml

# 国密 Truora（SM2）
truora/deploy/docker-compose-sm2.yml
```

修改对应的 `truora/deploy/docker-compose-xxxx.yml` 文件

```
version: '3.7'
services:
  truora-server:
    volumes:
      .......
      .......
      #- /sdk_path_of_chain_2/:/dist/conf/cert/2
```

在 `volumes` 配置中添加一行。

冒号 **：** 前表示新链的证书目录。

冒号 **：** 后表示 **容器中的证书路径**，Truora-Service 服务启动时，会从容器中的此目录加载证书。

修改连接配置

在部署时生成的连接配置文件，根据加密方式不同，生成的 **连接配置** 文件不同：

```
# 非国密 Truora（ECDSA）
truora/deploy/truora-ecdsa.yml

# 国密 Truora（SM2）
truora/deploy/truora-sm2.yml
```

修改对应的 `truora/deploy/truora-xxxx.yml` 文件。

注意： 在添加新链的证书配置时，`classpath` 冒号 **：** 的路径是 **容器中的证书路径** 相对 `/dist/conf` 的相对路径。

```
# 配置连接
group-channel-connections-configs:
  configs:
    ........
    ........
   ## 第二条链的连接信息，证书，群组列表以及对应的 IP:Port
    - chainId: 2
      caCert: classpath:cert/2/ca.crt
      sslCert: classpath:cert/2/node.crt
      sslKey: classpath:cert/2/node.key
      all-channel-connections:
        - group-id: 1
          connections-str:
```

(下页继续)

(续上页)

```
            - 127.0.0.1:20200

# 启用新链和群组
event:
  eventRegisters:
   ........
   ........
   - {chainId: 2, group: 1}
   #- {chainId: 2, group: 2}
```

详细配置修改，请参考：修改配置

提示

- *truora.yml* 配置文件中的 *${TRUORA_SERVICE_PORT:5021}* 表示 Truora-Service 在启动时，会优先从系统环境变量中读取 *TRUORA_SERVICE_PORT* 的值。如果 *TRUORA_SERVICE_PORT* 环境变量 **没有设置** 或者 **值为空**，则使用默认值 *5021*。

服务启停

如果一键部署脚本 `deploy_all.sh` 执行成功后显示 `Deploy Truora service SUCCESS!!`，表示部署成功。

- 使用 `bash start.sh` 启动 Truora 服务。
- 使用 `bash stop.sh` 停止服务。

在启动时，脚本会依次启动服务，并检测服务启动结果。

如果提示 `Truora service start up SUCCESS !!`，表示 Truora 服务启动成功。

如果启动失败，根据命令行的提示，检查启动失败服务的日志。关于查看服务的日志，请参考：日志查看

访问服务

Truora 服务启动成功后，打开浏览器，输入 `http://[IP]:5020`，比如：`http://127.0.0.1:5020`，访问 Truora-Web 服务，请参考：Truora-Web 服务介绍

提示

- 注意替换服务器的 *IP* 地址

关于 Truora 服务的 **开发教程**，请参考：Truora 开发教程

2.1.3 附录

部署工具源码

拉取部署工具远吗，需要使用 Git 工具。

```
# 初始化本地目录
git init Truora-Service && cd Truora-Service;

# 设置检出子目录 docker/deploy
```

(下页继续)

(续上页)

```
git config core.sparsecheckout true ;
echo "docker/deploy" >> .git/info/sparse-checkout ;

# 设置仓库地址，拉取部署工具
git remote add origin "https://github.com/WeBankBlockchain/Truora-Service.git";
git fetch --depth 1 && git checkout main

# 进入部署脚本目录
cd docker/deploy
```

查看日志

WeBASE-Front

WeBASE-Front 的日志位于相对目录 `webase/log/` 中。

查看 WeBASE-Front 日志：

```
# 服务启动日志
cat webase/log/WeBASE-Front.log

# 服务错误日志
cat webase/log/WeBASE-Front-error.log
```

MySQL

使用命令查看 MySQL 日志：

```
docker logs truora-mysql
```

FISCO-BCOS

FISCO-BCOS 的日志位于相对目录 `fiscobcos/nodes/127.0.0.1/node0/log` 中，注意替换 `node0` 目录。

查看 FISCO-BCOS 节点日志：

```
# 服务启动日志
#
# 注意替换 node0 目录
cat fiscobcos/nodes/127.0.0.1/node0/log/log_xxxxxxxxx.xx.log
```

Truora-Service

Truora-Service 的日志位于相对目录 `truora/deploy/log/server/` 中。

查看 Truora-Service 日志：

```
# 服务启动日志
cat truora/deploy/log/server/Oracle-Service.log

# 服务错误日志
cat truora/deploy/log/server/Oracle-Service-error.log
```

Truora-Web(Nginx)

Truora-Web 部署在一个 Nginx 的 Docker 容器中。

查看 Nginx 的日志：

```
docker logs truora-web
```

Truora-Web 的访问日志位于相对目录 `truora/deploy/log/nginx/oracle-access.log` 中。
Truora-Web 的错误日志位于相对目录 `truora/deploy/log/nginx/oracle-error.log` 中。

脚本参数

一键部署 脚本 `deploy_all.sh` 和 **独立部署** 脚本 `deploy_single.sh` 本质上都是通过调用 `util/deploy_util.sh` 脚本来完成部署。

区别

- *deploy_all.sh* 脚本执行时，自带 *-m*，*-w*，*-f* 参数。

```
# 查看部署脚本参数
bash util/deploy_util.sh -h

# 参数明细
Usage:
    deploy_util.sh [-k] [-m] [-w] [-f] [-M 3306] [-W 5002] [-B 5020] [-S 5021] [-
↪d] [-g] [-i fiscoorg] [-t] [-p] [-D] [-h]
    -k        Pull images from Docker hub.

    -m        Deploy a MySQL instance with Docker.
    -w        Deploy a WeBASE-Front service.
    -f        Deploy a 4 nodes FISCO-BCOS service.

    -M        Listen port of MySQL, default 3306.
    -W        Listen port of WeBASE-Front, default 5002.
    -B        Listen port of Truora-Web, default 5020.
    -S        Listen port of Truora-Service, default 5021.

    -d        Install dependencies during deployment.
    -g        Use guomi.

    -i        Organization of docker images, default fiscoorg.
    -t        Use [dev] tag for images of Truora-Service and Truora-Web. Only for␣
↪test.
    -p        Pull [dev] latest for images of Truora-Service and Truora-Web. Only␣
↪works when option [-t] is on.
    -D        Set log level of Truora to [ DEBUG ], default [ INFO ].

    -h        Show help info.
```

提示

- *-d* 参数安装的系统依赖包括：*OpenSSL*，*curl*，*wget*，*Docker*，*Docker Compose*
- *-i* 参数指定 Docker 镜像的组织名称。例如指定为: *testorg*，Docker 的拉取镜像命令就变成：*docker pull testorg/truora-service:${version}*
- *-p* 参数只有在启用 *-t* 参数后才有效

CDN 说明

由于 Docker 的镜像的大小一般偏大，访问国外时网络很不稳定，在下载大文件或者工具时，往往下载不下来。所以，针对部分工具，提供了 CDN 的下载地址。

重要:

- 部署工具包是 **'zip'** 文件
- Docker 镜像是 **'tar'** 文件

部署工具包

部署工具包用来部署 Truora 服务，同时支持：**一键部署** 和 **独立部署** 两种场景。

```
# 下载指定版本时替换 {VERSION} 版本号
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪deploy/docker-deploy-{VERSION}.zip" -O docker-deploy.zip
# 解压部署包
unzip docker-deploy.zip
```

版本号，从 https://github.com/WeBankBlockchain/Truora-Service/releases 获取。

比如，下载 v1.1.0 版：

```
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪deploy/docker-deploy-v1.1.0.zip" -O docker-deploy.zip
# 解压部署包
unzip docker-deploy.zip
```

Docker 镜像

Docker 镜像下载成功后，使用命令进行解压：

```
# 解压镜像包
docker load -i xxxx.tar

# 查看镜像版本
docker images -a
```

Truora

Truora 镜像包含两个服务的镜像：Truora-Service 和 Truora-Web。

```
## 下载指定版本时替换 {VERSION} 版本号

# 下载 Truora-Service
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/truora/truora-service-{VERSION}.tar" -O truora-service.tar
# 加载镜像
docker load -i truora-service.tar

# 下载 Truora-Web
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/truora/truora-web-{VERSION}.tar" -O truora-web.tar
# 加载镜像
docker load -i truora-web.tar
```

版本号，从 https://github.com/WeBankBlockchain/Truora-Service/releases 获取。

比如，下载 v1.1.0 版：

```
# 下载 Truora-Service
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/truora/truora-service-v1.1.0.tar" -O truora-service.tar
# 加载镜像
docker load -i truora-service.tar


# 下载 Truora-Web
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/truora/truora-web-v1.1.0.tar" -O truora-web.tar
# 加载镜像
docker load -i truora-web.tar
```

FISCO-BCOS

FISCO-BCOS 镜像是指 FISCO-BCOS 底层节点镜像，当前仅包含 `v2.6.0` 版本

```
# 下载 FISCO-BCOS v2.6.0 镜像
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/FISCO-BCOS/fiscobcos-v2.6.0.tar" -O fiscobcos.tar
# 加载镜像
docker load -i fiscobcos.tar
```

WeBASE-Front

WeBASE-Front 镜像是指 WeBASE 中间件中的子服务 WeBASE-Front 的镜像，当前仅包含 `v1.4.2` 版本

```
# 下载 WeBASE-Front v1.4.2 镜像
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/WeBASE/webase-front-v1.4.2.tar" -O webase-front.tar
# 加载镜像
docker load -i webase-front.tar
```

MySQL

MySQL 镜像是 Docker Hub 仓库中的官方 MySQL 镜像，当前仅包含 `5.7` 版本

```
# 下载 MySQL 5.7 镜像
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/official/mysql-5.7.tar" -O mysql.tar
# 加载镜像
docker load -i mysql.tar
```

Docker-Compose

Docker-Compose 镜像是 Docker Hub 仓库中的官方 Docker Compose 镜像，当前仅包含 `1.27.4` 版本

```
# 下载 Docker Compose 1.27.4 镜像
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/official/docker-compose-1.27.4.tar" -O docker-compose.tar
# 加载镜像
docker load -i docker-compose.tar
```

CHAPTER 3

版本升级

此文档仅适用于v2stable分支，不包括适配FISCO BCOS3.1.x+的分支

参见版本分支说明

Truora 在进行升级操作时，不同的部署方式，有不同的操作流程。

Truora 的升级的主要流程如下：

1. 数据库修改
2. Truora 版本更新
3. 更新 Truora 配置

提示

- 由于 **一键部署** 的目的主要是体验，开发和调试，所以 **不推荐** 对一键部署的预言机服务进行升级操作，推荐重新部署。
- 该升级教程，仅适用于采用 **独立部署** 的部署方式部署的 Truora 服务。

如果需要重新部署，请参考：一键部署

3.1 数据库

3.1.1 获取数据库脚本

在进行数据库升级时，需要优先确定当前的 Truora 版本号，根据版本号来判断需要执行的数据库脚本列表。

比如，当前 Truora 版本号为 `v1.0.0`，需要升级到版本 `v1.1.0`，只需要执行 `V2021.03.08__v1.1.0.sql` 脚本即可。

提示

- 需要执行的 SQL 是 **当前版本** 和需要升级到的 **目标版本** 之间所有的数据库脚本。

重要:

- 执行脚本时，需要要按照 **版本顺序依次** 执行。

关于如何获取当前版本，请参考：获取 Truora 服务版本

获取数据库的脚本地址：

- gitee 仓库
- GitHub 仓库

推荐使用 `gitee` 仓库。

3.1.2 执行数据库脚本

重要:

- 如果采用 **一键部署** 的方式部署，建议使用新版本的部署脚本，重新部署最新版本的服务。

采用独立部署的方式时，Truora 连接的数据库是开发者自行提供的数据库。

在获取数据库的的更新脚本后，需要用户连上 MySQL 数据库，根据版本号对比，依次执行数据库脚本中的 SQL 语句。

下面以从 `v1.0.0` 版本更新到 `v1.1.0` 版本为例：

- 下载数据库脚本

```
## 下载 v1.1.0 数据库更新脚本

# 从 gitee 下载
wget "https://gitee.com/WeBankBlockchain/Truora-Service/raw/main/src/main/
↪resources/db/migration/V2021.03.08__v1.1.0.sql"

# 从 GitHub 下载
wget "https://raw.githubusercontent.com/WeBankBlockchain/Truora-Service/dev/src/
↪main/resources/db/migration/V2021.03.08__v1.1.0.sql"
```

- 连接到 MySQL

```
# 连接 MySQL
$ mysql -u root -p
......

# 登陆成功后，切换到 Truora 数据库
mysql> use truora

# 执行数据库脚本
mysql> source V2021.03.08__v1.1.0.sql

......
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

Query OK, 0 rows affected (0.06 sec)
```

(下页继续)

(续上页)

```
Records: 0  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0
......
```

如果都是 `Query OK`，表示数据库脚本执行成功。

提示

- 如果升级时的版本跨越多个版本，请按照版本号，依次执行数据库脚本。

如果使用使用其它的工具，只要能连接到数据，然后依次数据数据库变更脚本即可。

3.2 Truora 服务

在更新完数据库后，需要更新 Truora-Service 和 Truora-Web 的 Docker 镜像版本。

3.2.1 修改 Truora 版本

- 从 CDN 获取 Truora 服务镜像，此处以升级到 `v1.1.0` 版本作为例：
  - 关于从 CDN 下载其它版本，请参考：下载 Truora CDN 镜像

```
# 此处以升级到 v1.1.0 为示范

## 下载 Truora-Service
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/truora/truora-service-v1.1.0.tar" -O truora-service.tar
# 加载镜像
docker load -i truora-service.tar

## 下载 Truora-Web
wget "https://osp-1257653870.cos.ap-guangzhou.myqcloud.com/WeBankBlockchain/Truora/
↪docker/truora/truora-web-v1.1.0.tar" -O truora-web.tar
# 加载镜像
docker load -i truora-web.tar
```

- 打开 Truora 服务的 docker-compose.yml 文件

```
# 进入 Truora 部署工具目录
$ cd /xxx/xxx/deploy

# 编辑 Truora 的 docker-compose.yml 文件
$ vim truora/deploy/docker-compose.yml
```

- Truora 版本号

更新 docker-compose.yml 文件中下面两行的版本为 `v1.1.0`

```
......

    image: fiscoorg/truora-web:v1.1.0

......

    image: fiscoorg/truora-service:v1.1.0
```

(下页继续)

(续上页)

```
......
```

3.2.2 修改 Truora 配置

- 打开 Truora 配置文件 truora.yml

```
# 进入 Truora 部署工具目录
$ cd /xxx/xxx/deploy

# 编辑 Truora 的 truora.yml 配置文件
$ vim truora/deploy/truora.yml
```

- 关闭 generate-ddl 功能

修改 truora.yml 文件中的配置为 false

```
#server config
......

spring:
......
  jpa:
    generate-ddl: false
......
```

- 重启 Truora 服务

```
# 进入 Truora 部署工具目录
$ cd /xxx/xxx/deploy

# 重启
$ bash stop.sh && bash start.sh
============================================================
Root dir: [/xxx/xxx/deploy]
String Truora..
Creating truora-service ... done
Creating truora-web     ... done
[INFO] Wait for Truora-Service start up on port:[5021]...
[INFO] Truora-Service start success.

[INFO] Wait for Truora-Web start up on port:[5020]...
[INFO] Truora-Web start SUCCESS.

Truora service start up SUCCESS !!
```

看到 Truora service start up SUCCESS 提示，表示重启成功，升级完成。

CHAPTER 4

Truora 开发教程

Truora 预言机服务中有两个角色：

- **Truora 服务运营方**

  服务运运营方需要部署 `Truora-Service` 和 `Truora-Web` 服务，并且部署预言机相关合约到链上，为预言机用户提供服务。

- **预言机用户**

  预言机用户需要根据自身业务，选择一个 Truora 服务运营方，并编写预言机合约（需要从服务运营方处获取预言机相关合约的地址），使用服务运营方提供的预言机服务。

4.1 开发流程

预言机服务开发的流程：

1. 获取 预言机 相关合约地址
   - 选择一个 Truora 服务运营方，并从运营方获取到 预言机 相关合约地址
   - 如果没有运营方，可以参考：安装部署 自行搭建 Truora 服务。部署完成后，可以通过 Truora-Web 获取 预言机 相关合约地址，请参考：查询系统合约地址
2. 开发合约
   - 编写，调试合约

4.2 开发 Truora 合约

4.2.1 获取链下 API 数据

用户可以参考 APISampleOracle.sol 合约实现自己的oracle业务合约。

默认支持`solidity0.6`版本合约。 `solidity0.4`在 `Truora-Service` 同级目录。合约解析如下：

- 用户合约需继承FiscoOracleClient合约

```
contract APISampleOracle is FiscoOracleClient
```

- 构造函数需要传入指定的Truora服务的 OracleCore合约 地址。地址可以通过前端界面或者后端接口获取。

```
  constructor(address oracleAddress) public {
        oracleCoreAddress = oracleAddress;
  }
```

- 设定自己要访问的url。修改url变量赋值即可，并且指定需要返回值类型。目前只支持单个返回值，返回值可以是 string,int256,bytes三种类型。调用request需要指定返回值类型，默认类型是 int256，因为solidity不支持浮点数，返回 int256 类型需要指定放大倍数 timesAmount。如果返回值是string,请参考APISampleOracleReturnString.sol合约。

```
  function request() public returns (bytes32)
    {
      // default return type is INT256
      //returnType = ReturnType.STRING;


      // Set your URL
      // url = "plain(https://www.random.org/integers/?num=100&min=1&max=100&
↪col=1&base=10&format=plain&rnd=new)";
         url = "json(https://api.exchangerate-api.com/v4/latest/CNY).rates.JPY";
         bytes32  requestId = oracleQuery(oracleCoreAddress, url, timesAmount);
         validIds[requestId] = true;
         return requestId;

    }
```

- 必须实现 **__callback(bytes32 _requestId, bytes memory _result)** 方法，用于Truora-Service服务回调获取的结果。
- **get()** 方法获取本次请求结果, 可自行修改此函数, 获取结果后进行自己业务逻辑的计算。

URL格式规范

目前支持json和text/plain两种访问格式。并且链下API的url建议支持HTTPS访问(安全因素考虑)。 遵循jsonpath格式，子元素 用 “.” 表示,数组用 “[]”表示。

text/plain默认取第一行，也可指定数组下标取特定行。 jsonpath规范可以参考 jsonpath

//获取链下随机数API plain(https://www.random.org/integers/?num=100&min=1&max=100&col=1&base=10&format=plain&rnd=new)

//获取人民币对日元汇率API json(https://api.exchangerate-api.com/v4/latest/CNY).rates.JPY

// 查询某城市某天最高温度 json(https://devapi.qweather.com/v7/weather/3d?location=101280601&key=90d8a8ee98ff495694dce72e96f53a18).daily{[}1{]}.tempMax

---

4.2.2 获取VRF随机数

用户可以参考 RandomNumberSampleVRF.sol 合约实现自己的oracle业务合约, 默认支持solidity0.6版本合约。 solidity0.4 在 Truora-Service 同级目录。合约解析如下:

- 用户合约需继承 VRFClient 合约

```
contract RandomNumberSampleVRF is VRFClient
```

- 构造函数需要传入指定的Truora服务方的 VRFCore合约地址和公钥哈希值。地址和哈希值都可以通过前端界面或者后端接口获取。

```
constructor(address _vrfCore, bytes32 _keyHash) public {
       vrfCoreAddress = _vrfCore;
       keyHash = _keyHash;
   }
```

- 设定自己提供的随机数种子值。

```
function getRandomNumber(uint256 userProvidedSeed) public returns (bytes32 ) {
     bytes32  requestId =  vrfQuery(vrfCoreAddress, keyHash, userProvidedSeed);
     validIds[requestId] = true;
     return requestId;
 }
```

- 必须实现 **__callbackRandomness(bytes32 requestId, uint256 randomness)** 方法，用于Truora-Service服务回调获取的结果。
- **get()** 方法获取本次随机数请求结果, 可自行修改此函数, 获取结果后进行自己业务逻辑的计算。

4.3 业务合约参考

下面以一个简单抽奖合约为例，介绍下一个简单抽奖业务怎么使用 Truora 预言机合约。

抽奖合约LotteryOracle.sol 实现了一个简单的抽奖逻辑， 通过使用上述APISampleOracle.sol 获取随机数结果。请保证 APISampleOracle 合约的url是获取获取随机数的url。 默认支持solidity0.6版本合约。 solidity0.4自行修改合约第一行的编译器版本即可。合约解析如下：

- 构造函数需要传入获取随机数合约 APISampleOracle 地址。

```
contract LotteryOracle {

 enum LOTTERY_STATE { OPEN, CLOSED }
 LOTTERY_STATE public lottery_state;
 address[] public players;
 uint256 public lotteryId;
 APISampleOracle private oracle;
 bytes32  private requestId;
 event Winner(uint256  lotteryId, address winner ,int256 ramdomness);


 constructor(address randomOracle) public {
     oracle = APISampleOracle(randomOracle);
     lotteryId = 0;
     lottery_state = LOTTERY_STATE.CLOSED;
 }
```

- 开始抽奖函数需要传入参与者的地址。简单状态校验后，然后通过调用 APISampleOracle 的 request 函数获取随机数。

```
   function start_new_lottery(address[] memory _players) public {
         require(lottery_state == LOTTERY_STATE.CLOSED, "can't start a new␣
↪lottery yet");
         lottery_state = LOTTERY_STATE.OPEN;
         players = _players;
         lotteryId++;
         requestId = oracle.request();
     }
```

- 获取抽奖结果函数回返回中奖者地址。pickWinner 函数获取随机数结果，并对总参与人数取余，得出中奖者地址。

```
function pickWinner() public returns(address) {
        require(oracle.checkIdFulfilled(requestId) == false, " oracle query has not been fulfilled!");

        int256 randomness  = oracle.getById(requestId);
        uint256 index = uint256(randomness) % players.length;
        address winner = players[index];
        players = new address[](0);
        lottery_state = LOTTERY_STATE.CLOSED;
        emit Winner(lotteryId, winner, randomness);
        return winner;
    }
```

V1.1.0版本已加入通过VRF产生链上安全可验证随机数方案，用户也可参考 LotteryOracleUseVrf.sol 抽奖逻辑大部分相同，只是获取随机数获取方式从 api 方式改成 vrf 方式。

4.4 fiscoOracleClient 合约解析

- 抽象合约，__callback方法待实现。

```
function __callback(bytes32 requestId, int256 result) public {}
```

- 发起oracle请求，oracleQuery 函数会传入相关参数并调用 oracleCore 合约的 query方法。

```
  function oracleQuery(uint expiryTime, string memory datasource, address _oracle, string memory url, uint256 timesAmount, bool needProof) internal
  returns (bytes32 requestId) {
    // calculate the id;
    oracle = OracleCoreInterface(_oracle);
    int256 chainId;
    int256 groupId;
    ( chainId, groupId) = oracle.getChainIdAndGroupId();
    requestId = keccak256(abi.encodePacked(chainId, groupId, this, requestCount));
    pendingRequests[requestId] = _oracle;
    emit Requested(requestId);

    require(oracle.query(address(this),requestCount, url,timesAmount, expiryTime, needProof),"oracle-core invoke failed!");
    requestCount++;
    reqc[msg.sender]++;

    return requestId;
  }
```

4.5 VRFClient 合约解析

- 抽象合约，__callback方法待实现。

```
function __callbackRandomness(bytes32 requestId, uint256 randomness) internal virtual;
```

- 发起oracle随机数请求，vrfQuery 函数会传入相关参数并调用 VRFCore 合约的 randomnessRequest方法。为了保证用户提供的种子足够随机，randomnessRequest函数会把用户种子 （_consumerSeed），预言机服务方公钥哈希 （_keyHash），用户合约地址

（_sender），用户合约发送请求次数（nonce），区块哈希（blockhash）一起做哈希处理得出最终VRF随机数种子。

```
function randomnessRequest(
    bytes32 _keyHash,
    uint256 _consumerSeed,
    address _sender) external returns(bool) {
    // record nonce
    uint256 nonce = nonces[_keyHash][_sender];
    // preseed
    uint256 preSeed = makeVRFInputSeed( _keyHash, _consumerSeed, _sender, nonce);

    bytes32 requestId = makeRequestId(chainId, groupId, _keyHash, preSeed);
    // Cryptographically guaranteed by preSeed including an increasing nonce
    assert(callbacks[requestId].callbackContract == address(0));
    callbacks[requestId].callbackContract = _sender;
    callbacks[requestId].seedAndBlockNum = keccak256(abi.encodePacked(
        preSeed, block.number));
    emit RandomnessRequest(address (this), _keyHash, preSeed, block.number,
      _sender, requestId, callbacks[requestId].seedAndBlockNum, _consumerSeed);
    nonces[_keyHash][_sender] = nonces[_keyHash][_sender].add(1);
    return true;
  }
```

- VRF随机数验证逻辑可以参考 VRFCore的getRandomnessFromProof 方法。

4.6 开发示例

4.6.1 部署预言机服务

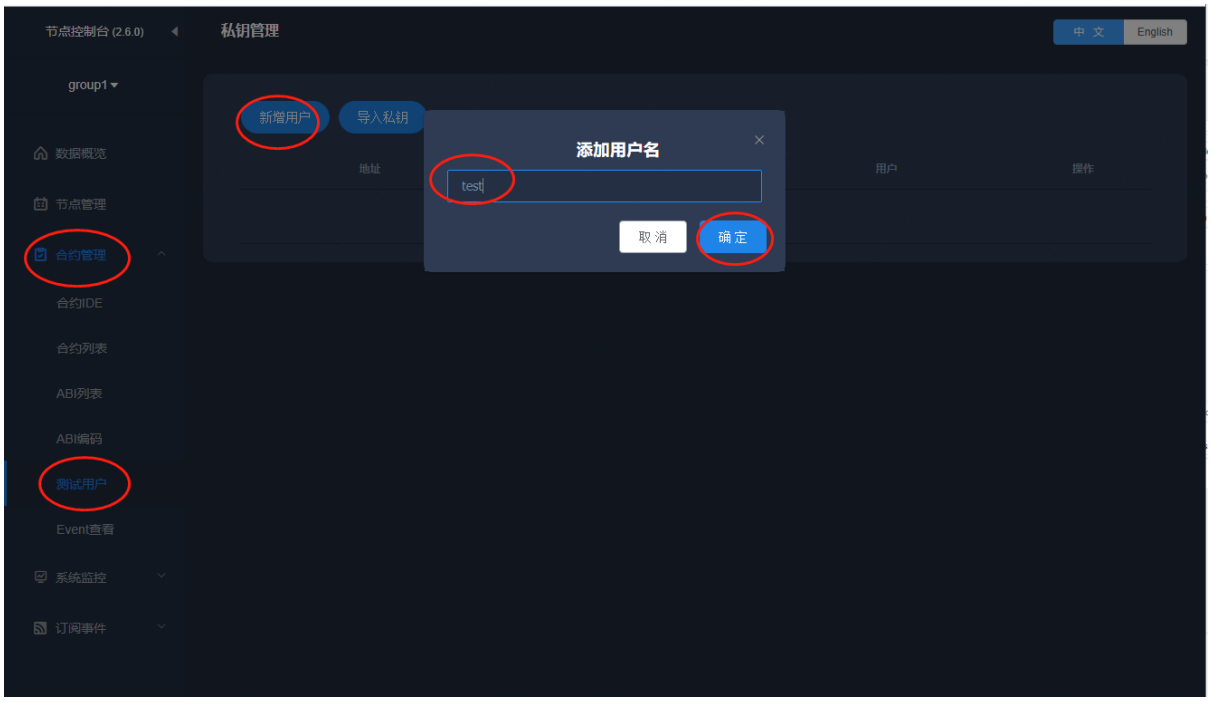
部署 Truora 服务，示例使用 **一键部署**，部署整套开发，调试环境，请参考：安装部署。

4.6.2 获取链下 API 数据

编写预言机合约

打开一键部署的 WeBASE-Front 页面，默认：http://{IP}:5002/WeBASE-Front/，使用部署主机的 IP 地址替换 {IP}。

- 点击左边 **合约管理 -> 测试用户**，创建一个调试用户 test

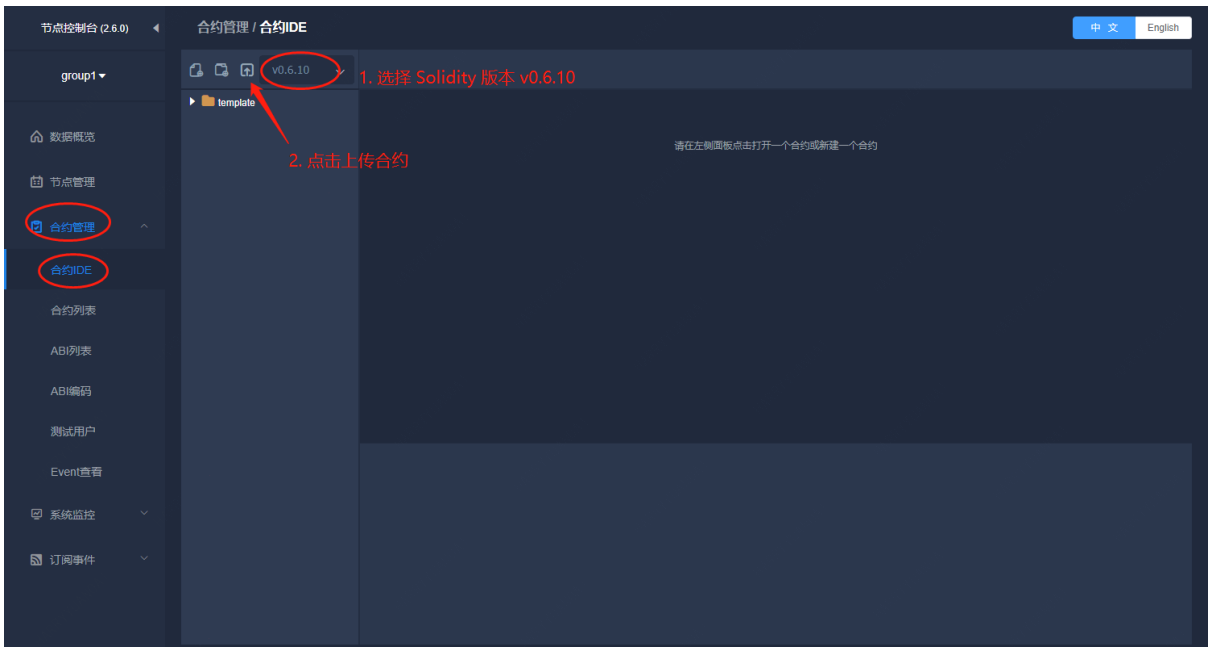


- 点击左边 **合约管理 -> 合约 IDE**，选择 `solidity` 版本，上传 Oracle 相关合约，包括以下几个合约：

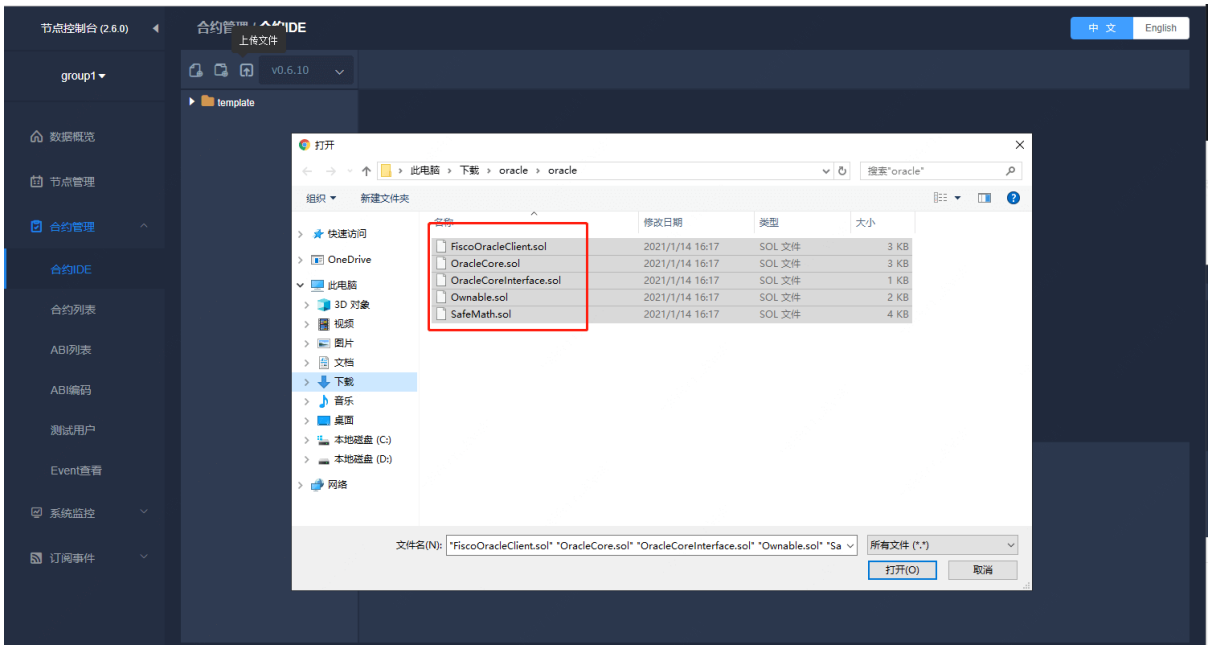
```
FiscoOracleClient.sol
OracleCore.sol
OracleCoreInterface.sol
Ownable.sol
SafeMath.sol
```

gitee 仓库：Oracle 相关合约目录

GitHub 仓库：Oracle 相关合约目录



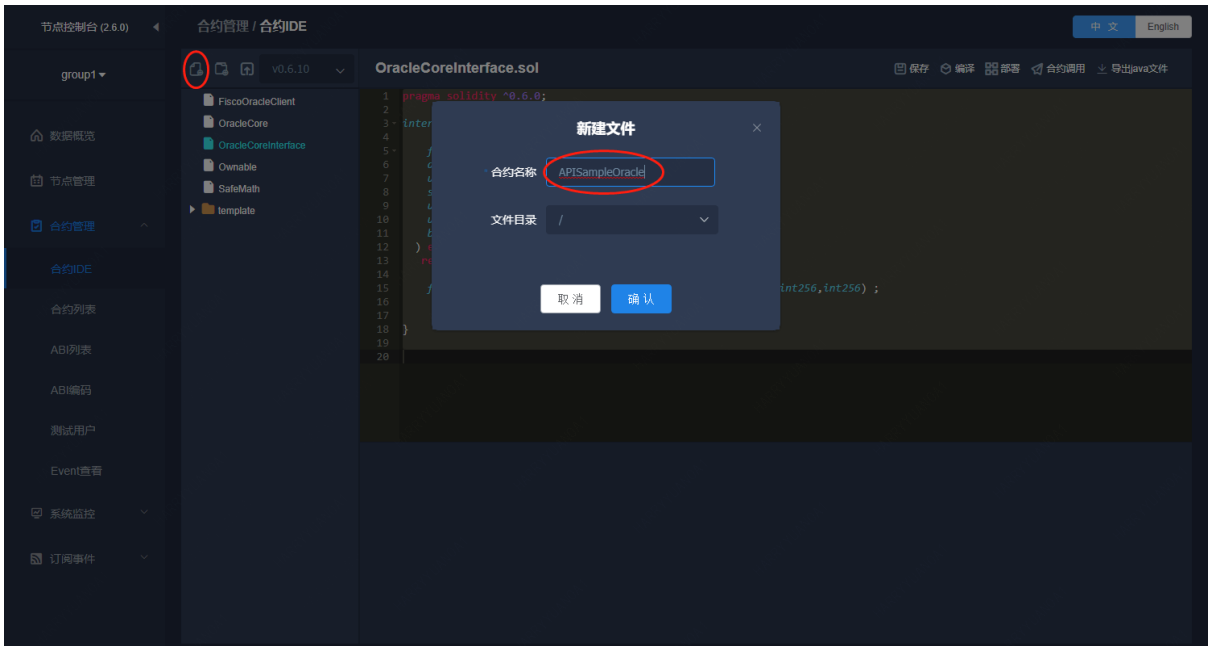
- 确认后，选择上传目录，此处选择根目录 /



重要:

- 注意需要上传 **五个** 合约文件。

- 在 **合约 IDE** 中，创建一个 `APISampleOracle` 合约，继承 `FiscoOracleClient` 合约，如下

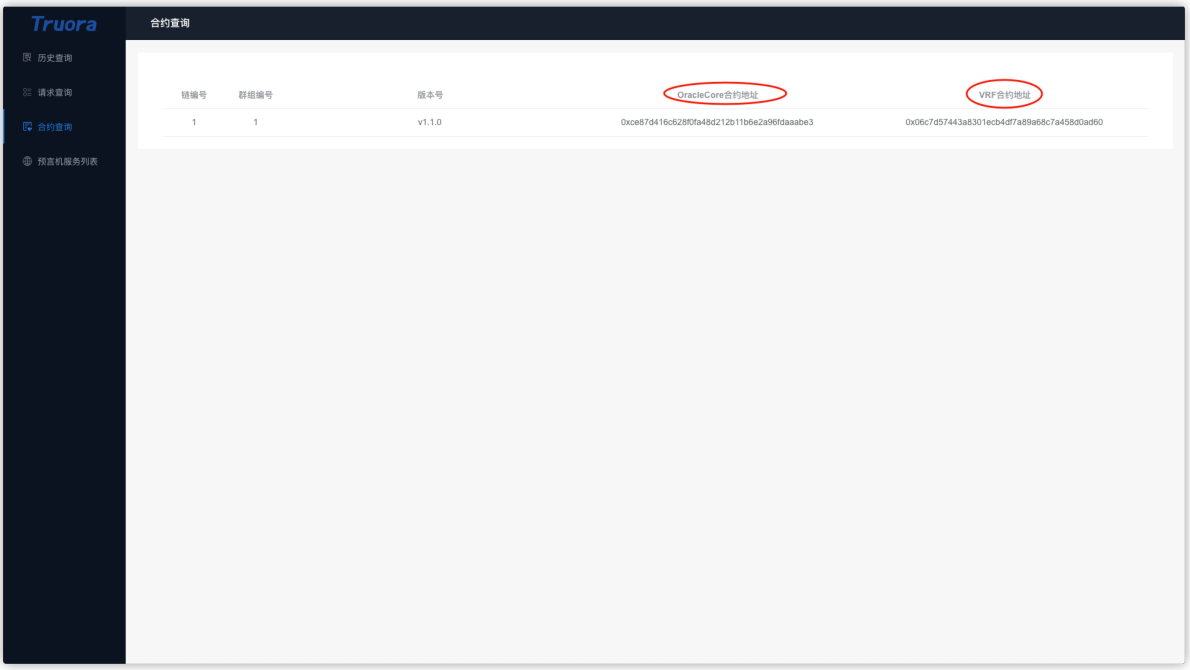


APISampleOracle 合约的代码，请参考：

gitee: APISampleOracle.sol 合约GitHub: APISampleOracle.sol 合约

获取合约地址

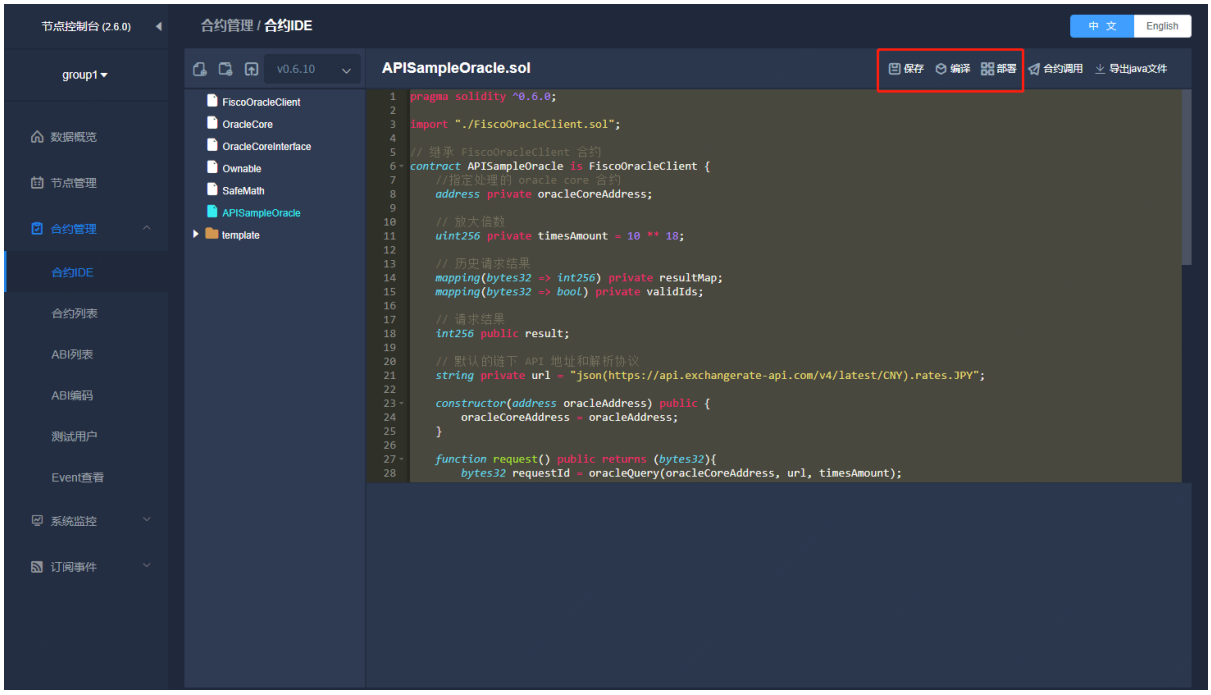
在部署 `APISampleOracle` 时，需要获取 `OracleCore` 合约地址，可以通过 Truora-Web 查看。



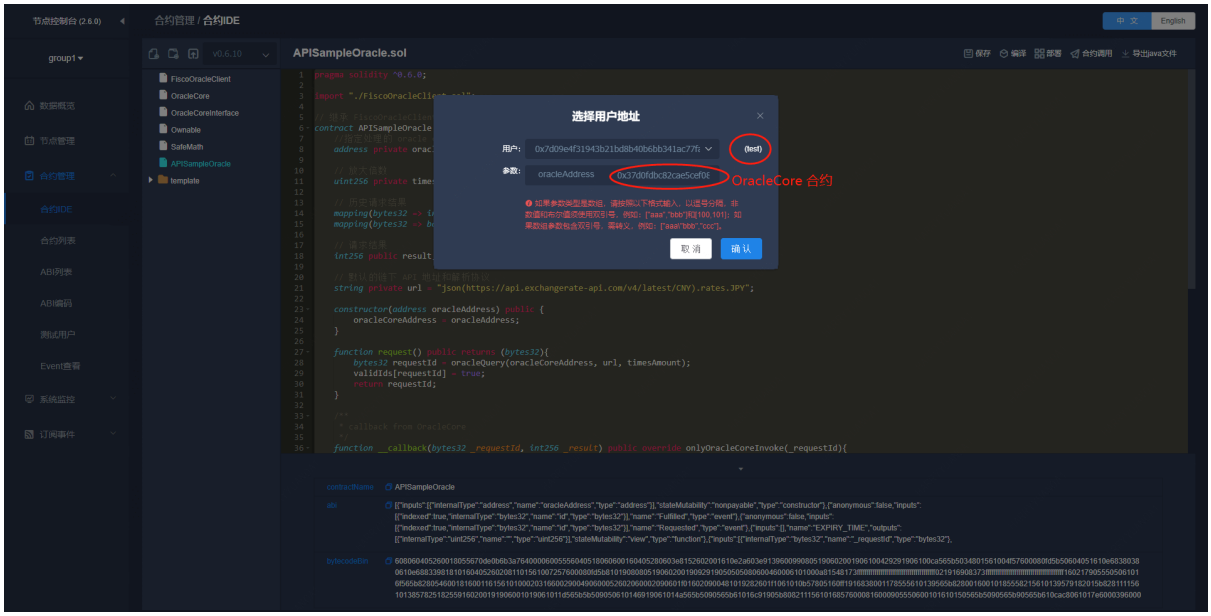
如果需要使用 RESTful 接口获取，请参考：OracleCore 合约地址查询接口

部署合约

选择 APISampleOracle 合约文件，依次点击 **保存** -> **编译** 编译合约。

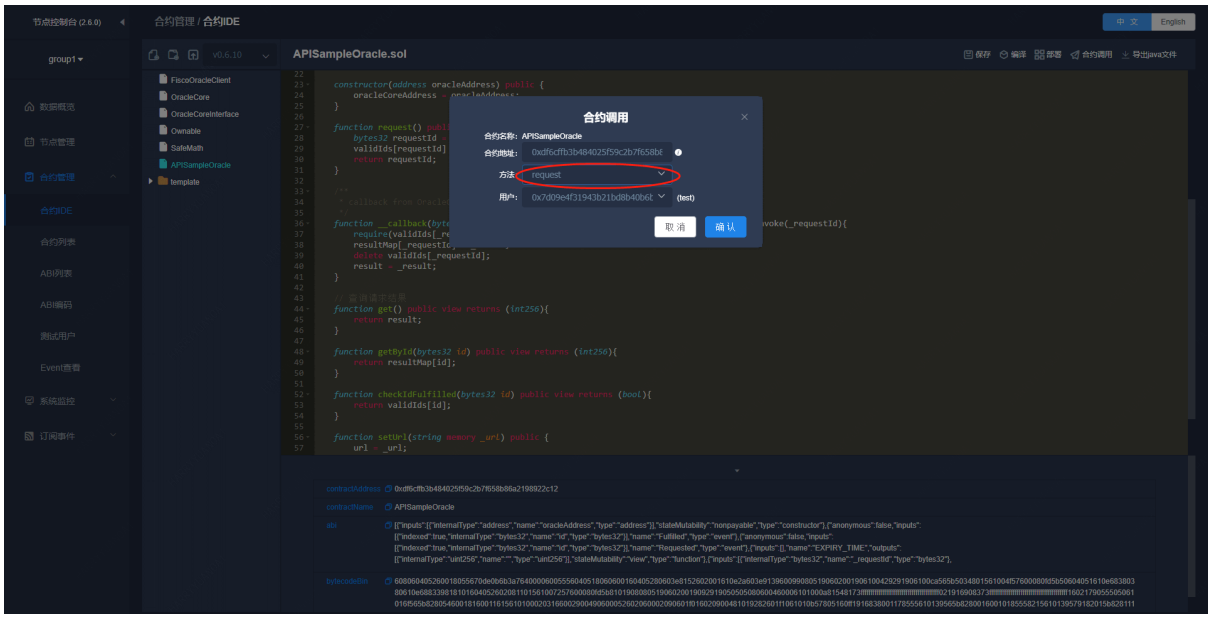


点击 **部署** 按钮，部署 APISampleOracle 合约，选择刚刚创建的测试用户 test，输入 OracleCore 合约地址：

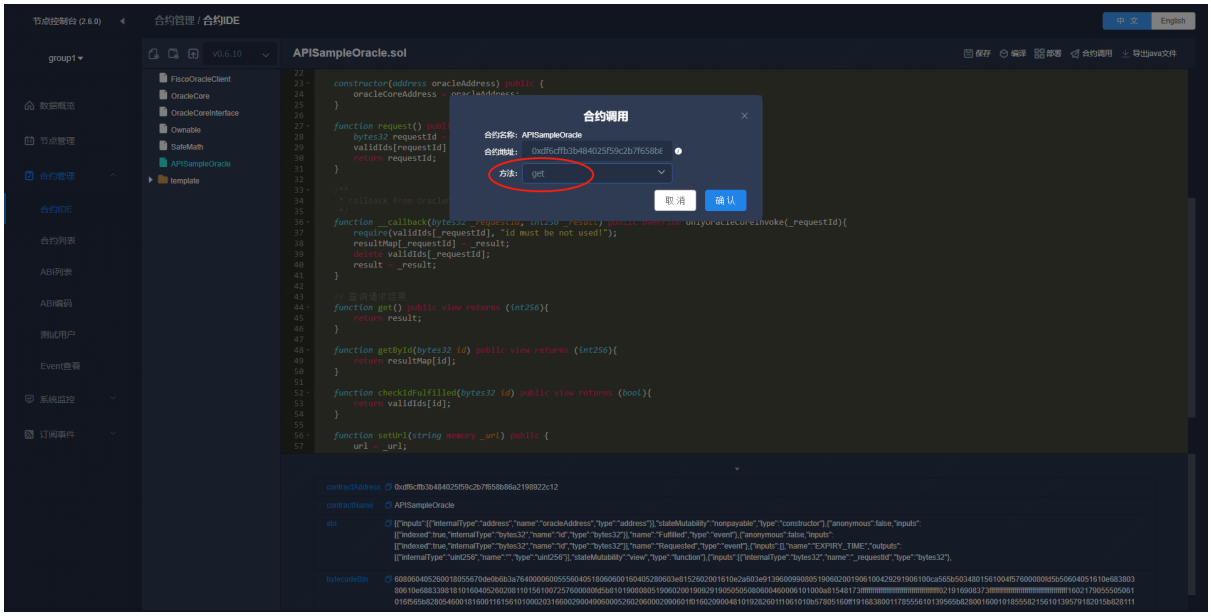


合约调用

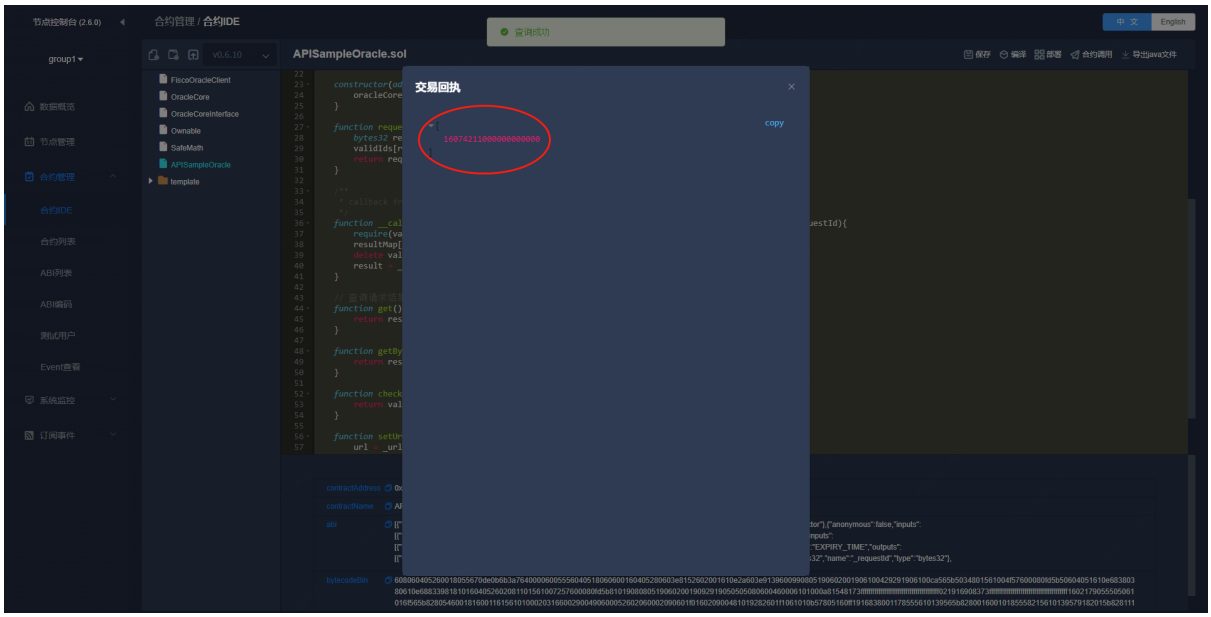
调用 `APISampleOracle` 合约的 `request` 方法，触发预言机获取数据



调用 `APISampleOracle` 合约的 `get` 方法，查看预言机返回的结果



结果显示如下，此处是获取日元到人民币的汇率，放大 10^18 倍的结果：

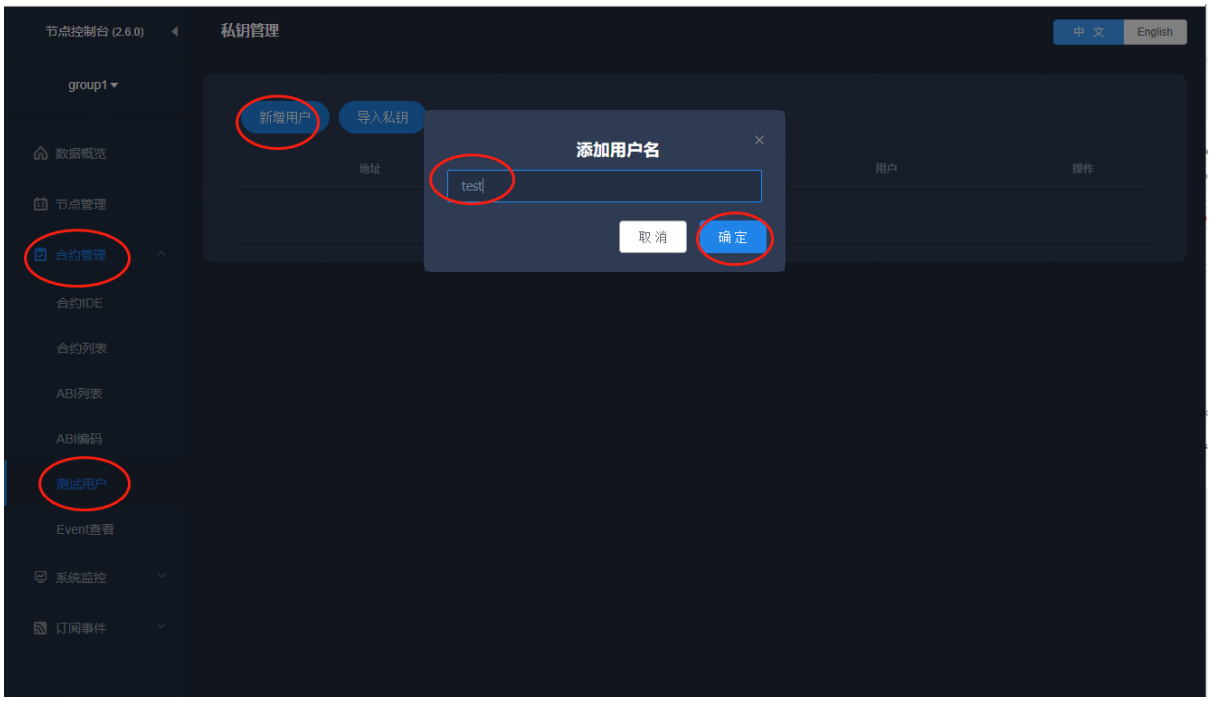


4.6.3 获取 VRF 随机数

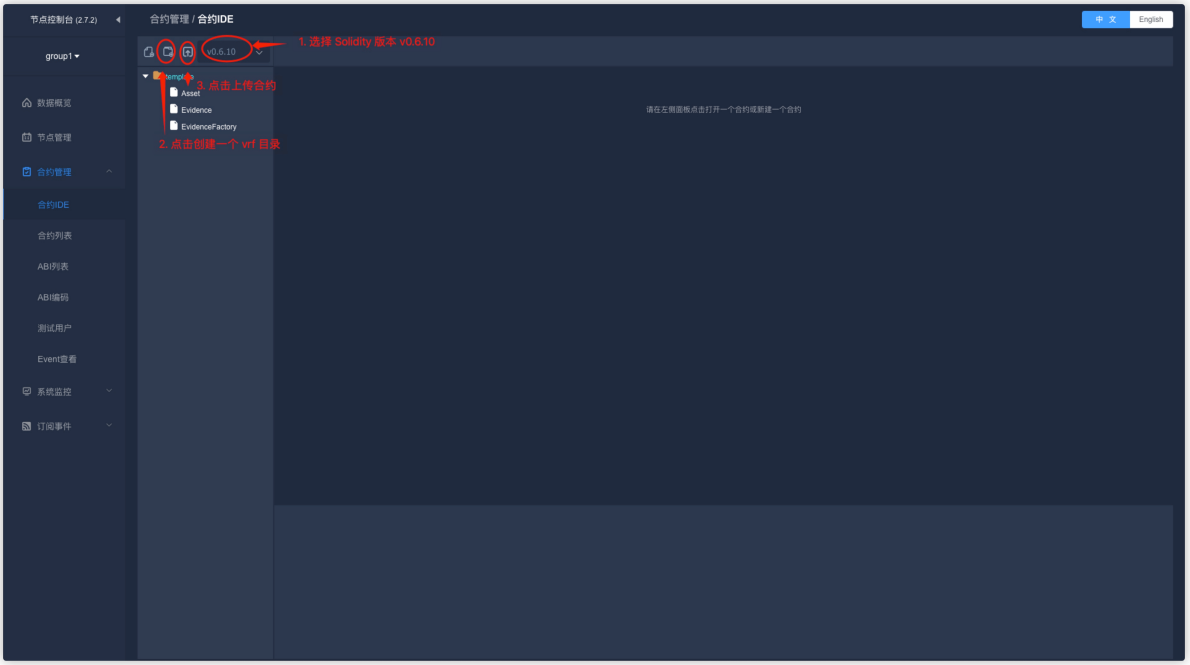
编写预言机合约

打开一键部署的 WeBASE-Front 页面，默认：`http://{IP}:5002/WeBASE-Front/`，使用部署主机的 IP 地址替换 `{IP}`。

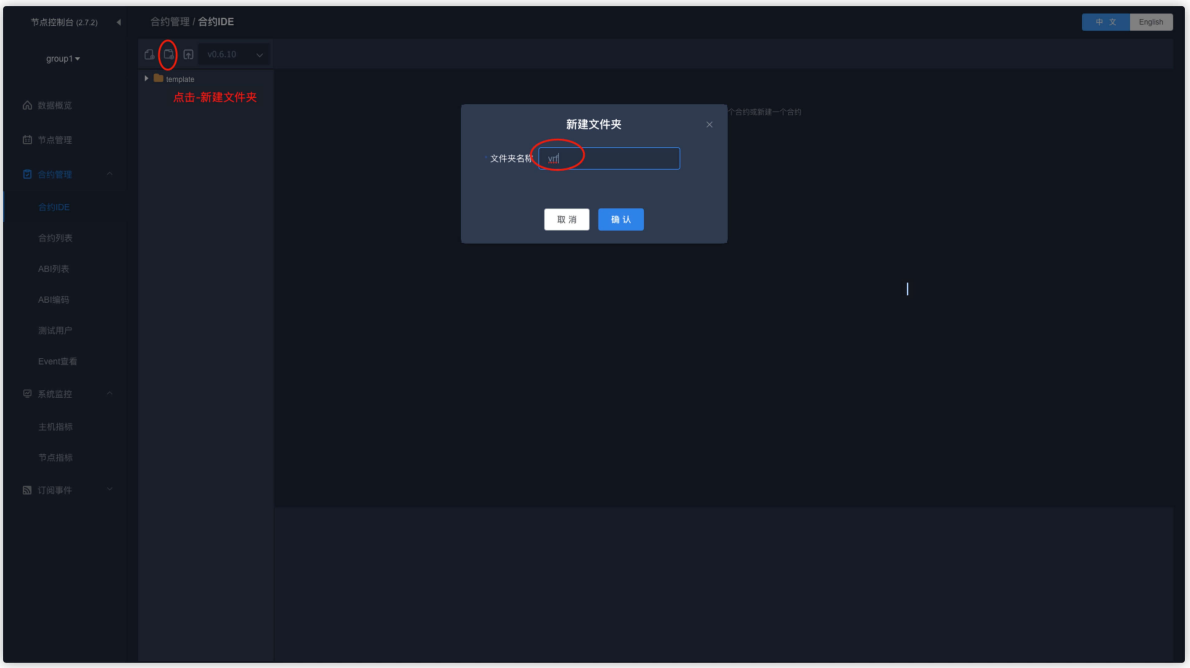
- 点击左边 **合约管理 -> 测试用户**，创建一个调试用户 `test`



- 点击左边 **合约管理 -> 合约 IDE**，选择切换 `solidity` 版本



- 创建一个 `vrf` 目录，点击确认



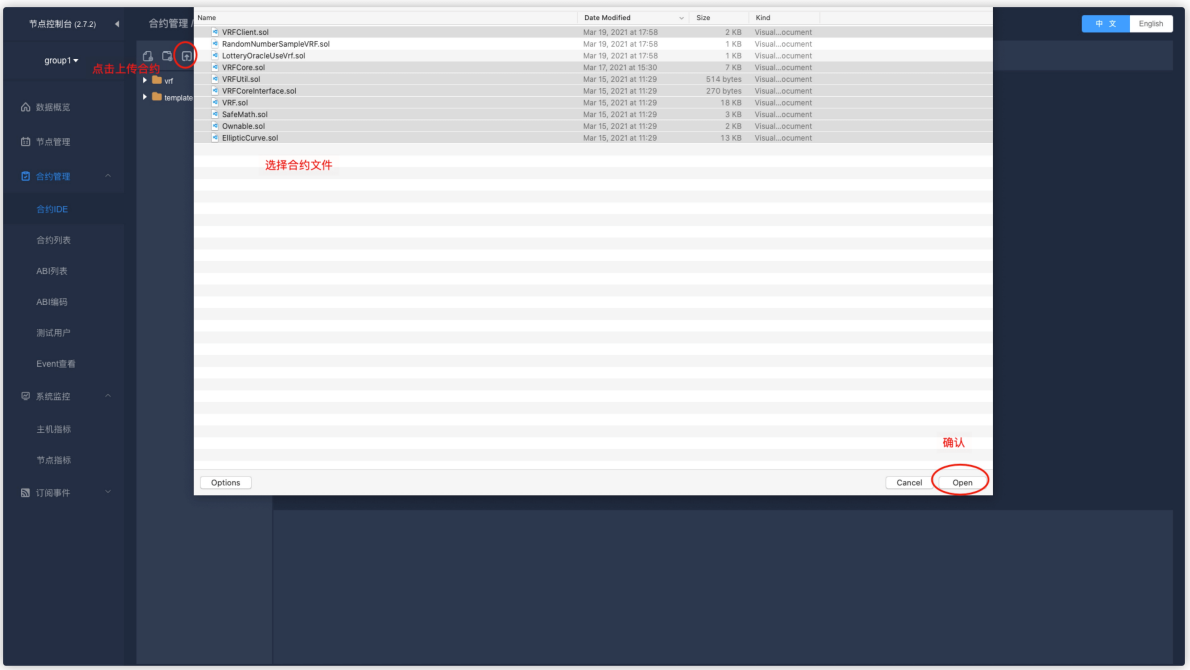
- 上传 VRF 相关合约，包括以下几个合约：

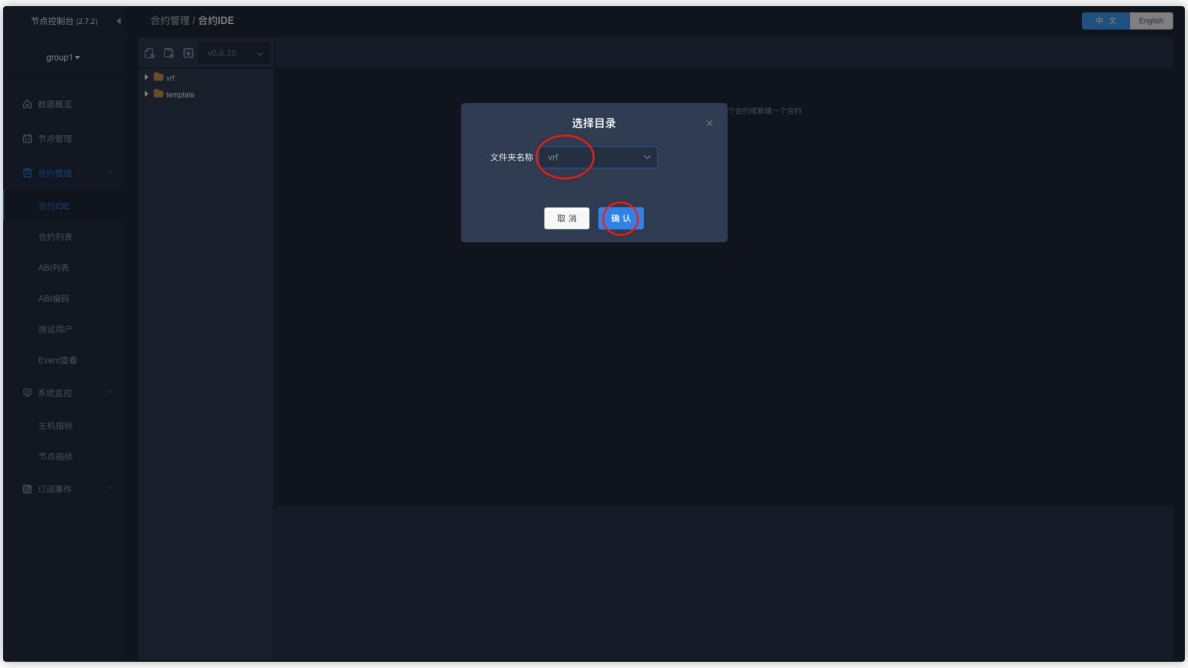
```
EllipticCurve.sol
Ownable.sol
SafeMath.sol
VRF.sol
VRFClient.sol
VRFCore.sol
VRFCoreInterface.sol
VRFUtil.sol
```

gitee 仓库：VRF 相关合约目录

GitHub 仓库：VRF 相关合约目录

- 确认后，选择上传目录，此处选择根目录 `/vrf`

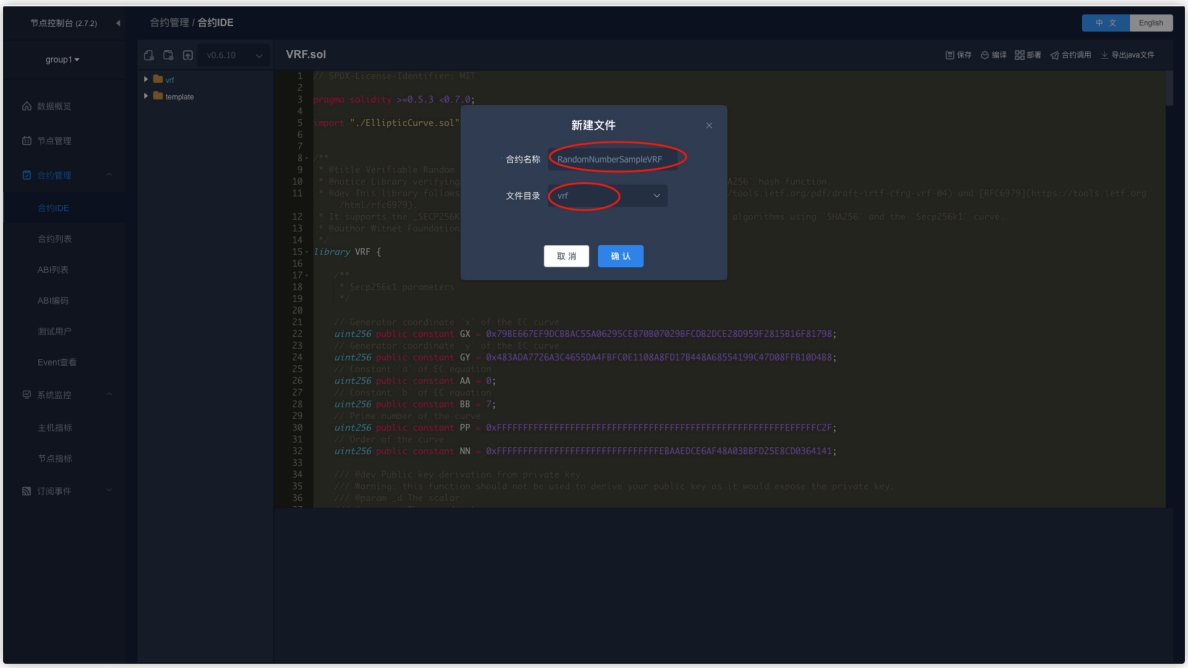




重要:

- 注意需要上传 **八个** 合约文件。

- 在 **合约 IDE** 中，创建一个 `RandomNumberSampleVRF` 合约，选中 `vrf` 目录，继承 `VRFClient` 合约，如下



具体代码，请参考：

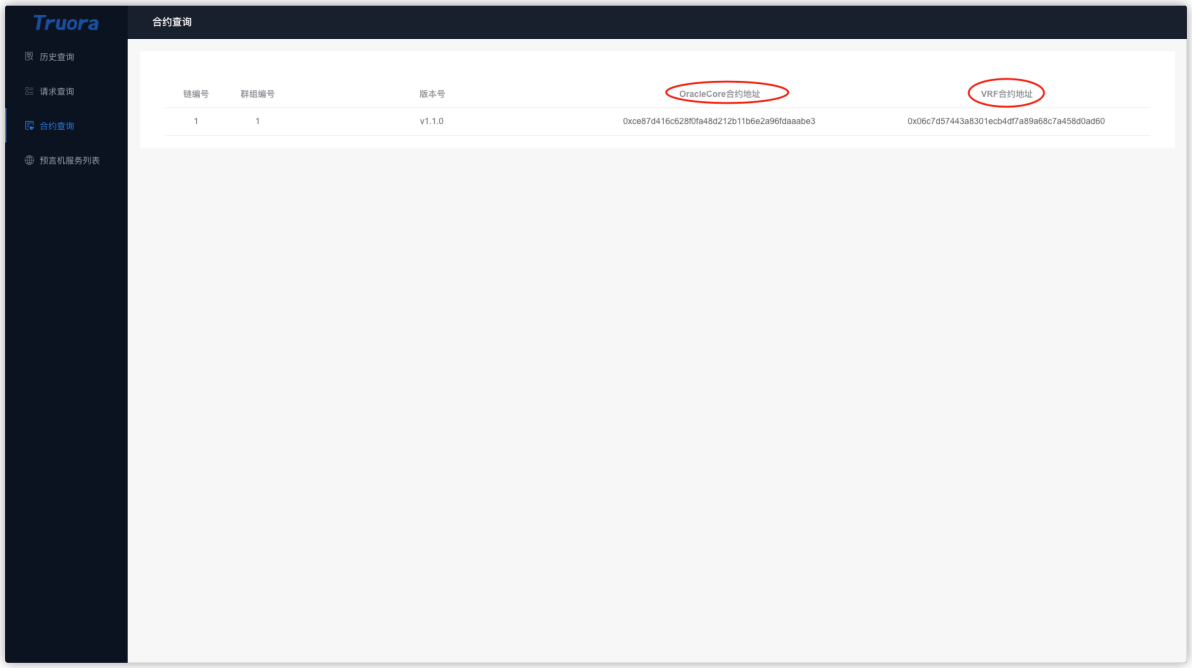
gitee: RandomNumberSampleVRF.sol 合约

GitHub: RandomNumberSampleVRF.sol 合约

获取合约地址和 _keyHash

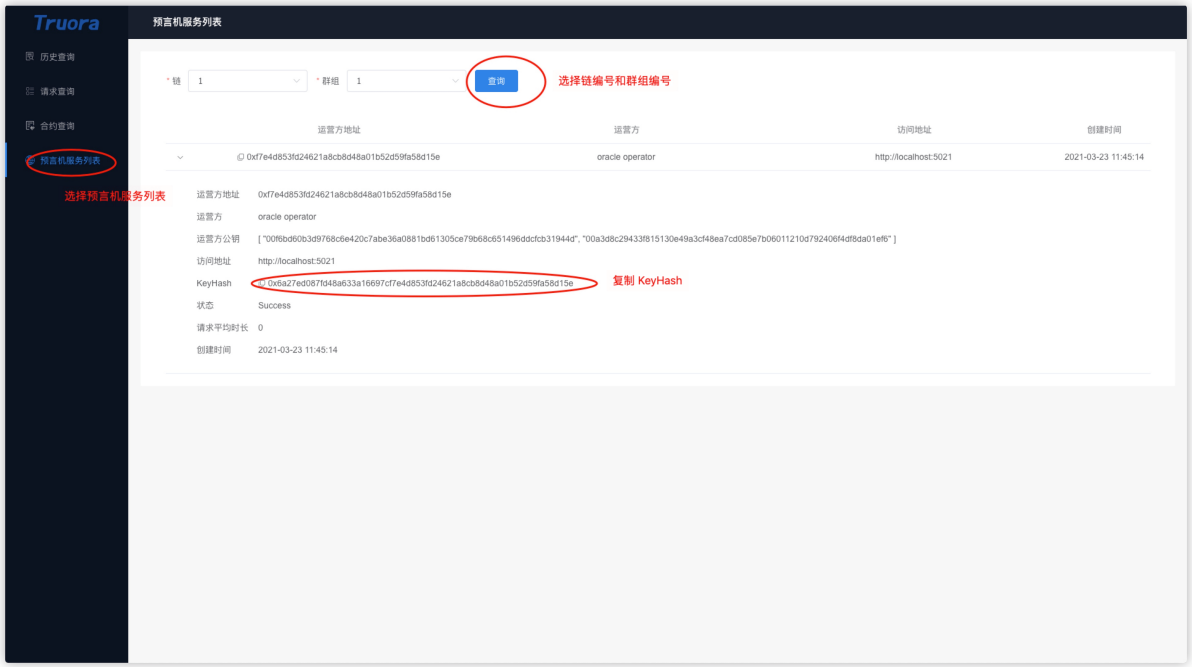
- 获取 VRF 合约地址

在部署 RandomNumberSampleVRF 时，需要获取 VRF 合约地址，可以通过 Truora-Web 查看。



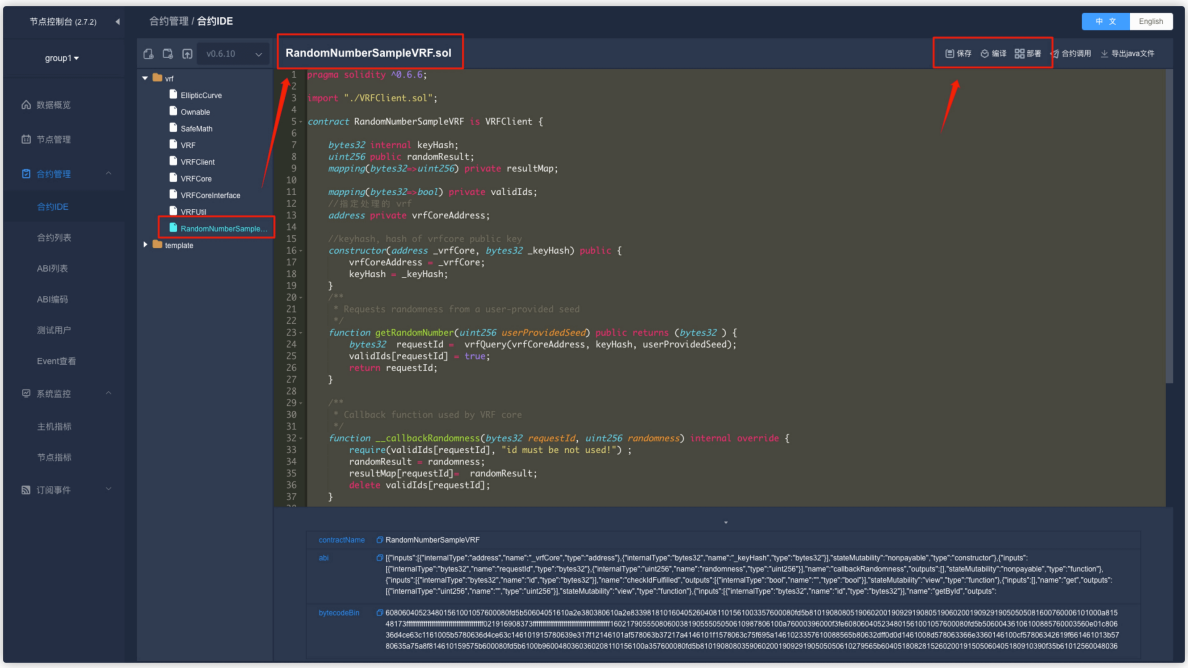
如果需要使用 RESTful 接口获取，请参考：VRFCore 合约地址查询接口

- 获取 _keyHash

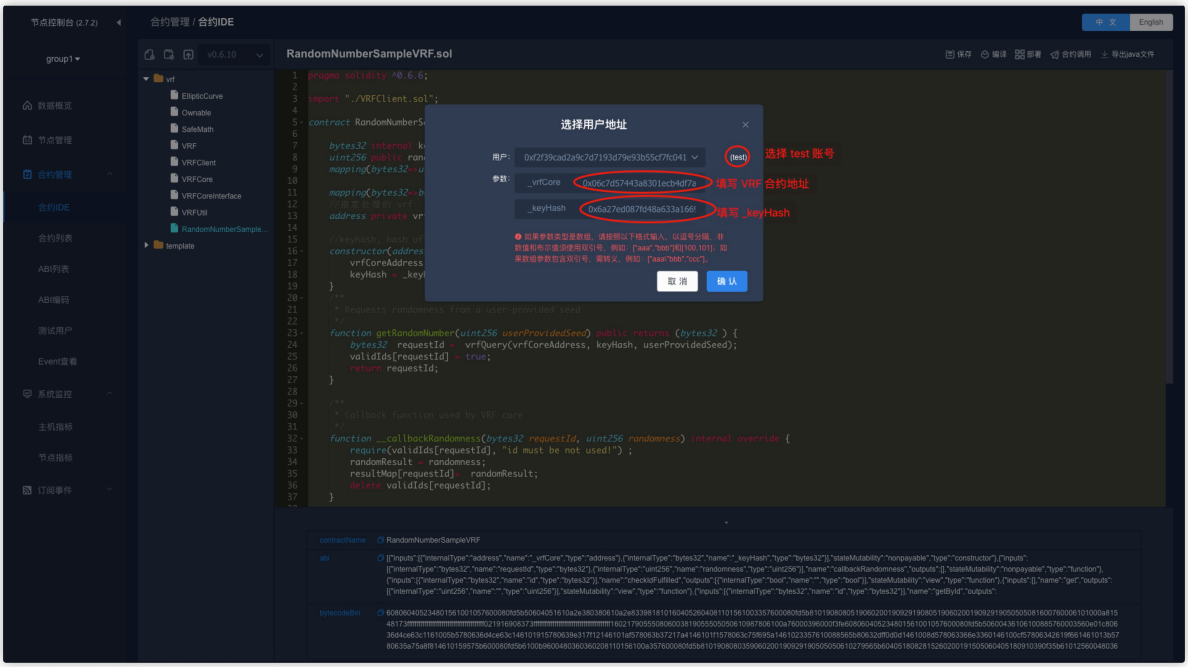


部署合约

选择 RandomNumberSampleVRF 合约文件，依次点击 **保存** -> **编译** 编译合约。

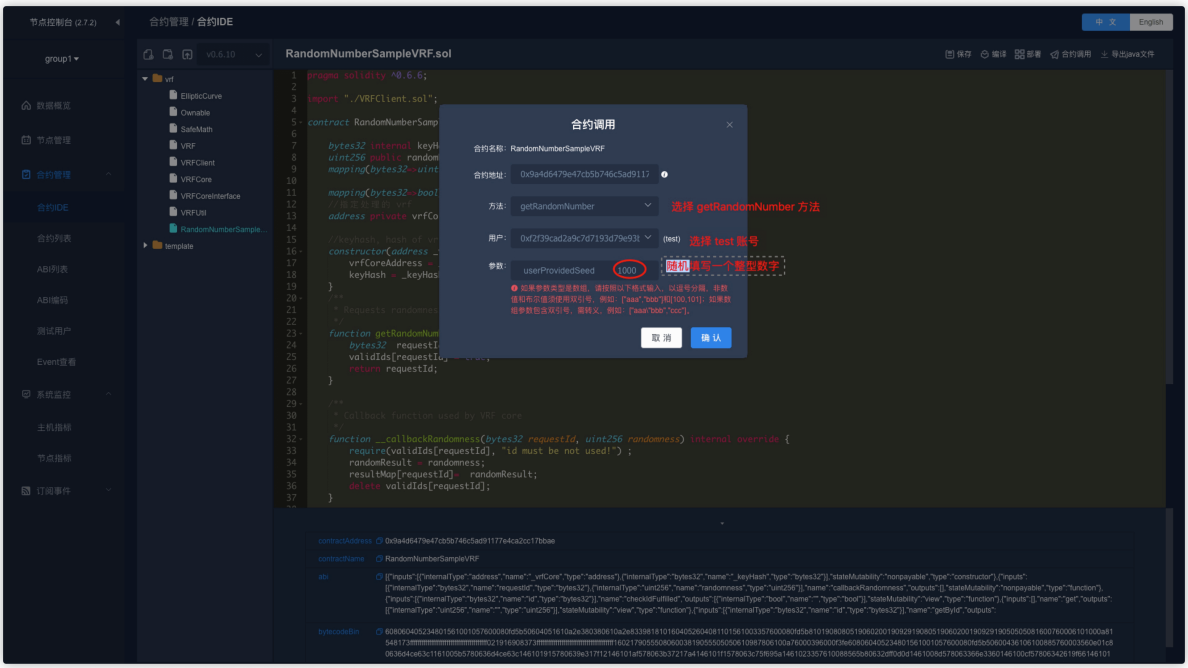


点击 **部署** 按钮，部署 `RandomNumberSampleVRF` 合约，选择刚刚创建的测试用户 `test`，输入 `OracleCore` 合约地址：

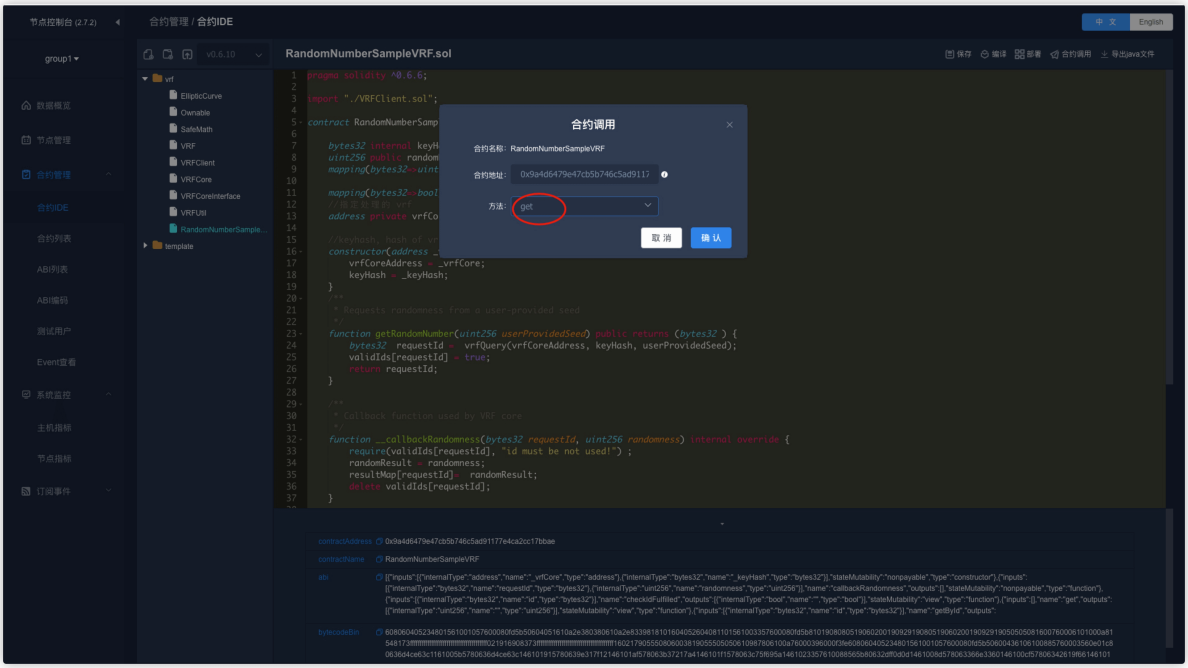


合约调用

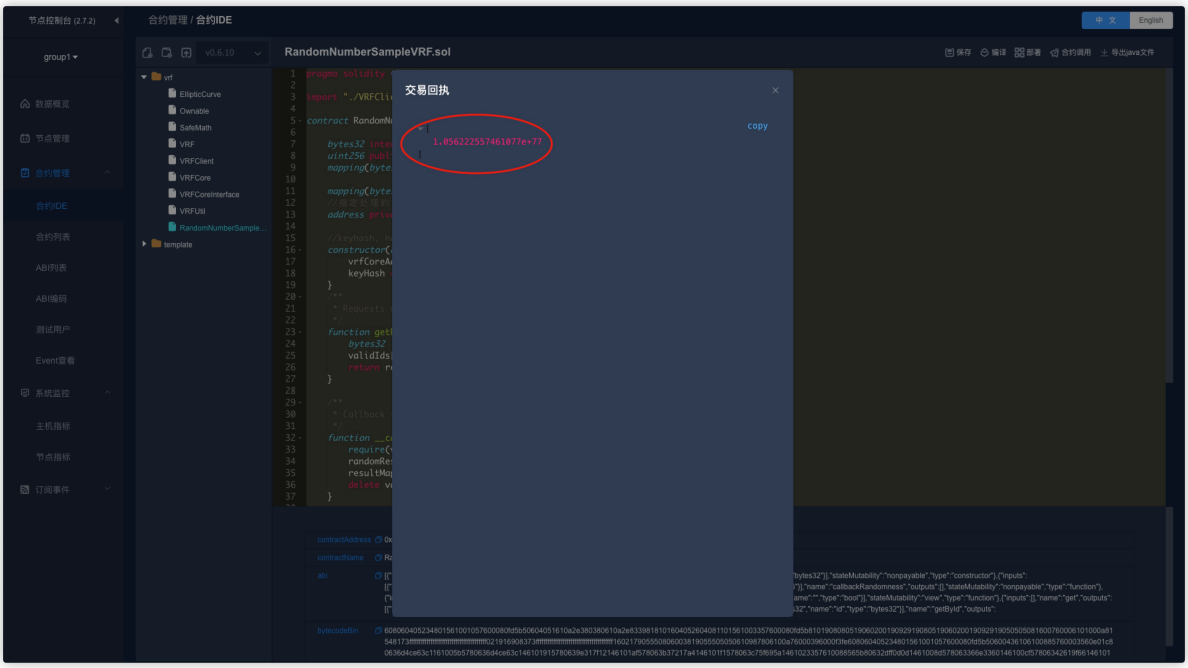
调用 `RandomNumberSampleVRF` 合约的 `getRandomNumber` 方法，`userProvidedSeed` 处填写任意一个随机的整型数字即可，点击 **确认** 触发预言机生成随机数



调用 RandomNumberSampleVRF 合约的 get 方法，查看预言机返回的随机数结果



结果显示如下，表示生成的链上随机数结果。


节点控制台 (2.7.2)
合约管理 / 合约IDE
中 文
English
group1
RandomNumberSampleVRF.sol
保存
编译
部署
合约调用
导出java文件
数据概览
节点管理
合约管理
合约IDE
合约列表
ABI列表
ABI编码
测试用户
Event查看
系统监控
主机指标
节点指标
订阅事件
v0.6.10
vrf
EllipticCurve
Ownable
SafeMath
VRF
VRFClient
VRFCore
VRFCoreInterface
VRFUtil
RandomNumberSample...
template
交易回执
copy
1.0562225574610776+77
contractAddress
contractName
abi
bytecodeBin

CHAPTER 5

案例集锦

5.1 积分抽奖

本项目结合 Turora 实现一个区块链积分抽奖的Sample，以演示随机数的使用方法，给开发者做为参考实现。 有两种实现方案，基于 Turora 链下 API 方式获取随机数，以及基于 Turora 的VRF方式获取链下随机数。

体验此功能前，请阅读 Truora 开发教程

5.1.1 业务流程

1 主持人开始一轮抽奖，初始化此轮抽奖每个参与者需要的积分数量，以及初始化参与此次抽奖的用户（地址）；

2 参与抽奖的用户存入积分；

3 主持人关闭积分存入功能，并向预言机发起随机数请求；

4 主次人开奖。根据预言机获取的随机数，对参赛选手人数取余，得到中奖者的地址，将积分发送给中奖者。

5.1.2 合约函数说明

基于Truora API 方式的抽奖合约LotteryBacOracle

基于Truora VRF 方式的抽奖合约LotteryBacUseVrf

原理基本类似，我们以 Truora API 方式的抽奖合约作为讲解。

- 构造函数

  调用合约的构造函数部署抽奖合约，需要传入两个参数：

  - randomOracle: 已部署客户端的随机数合约的地址，用于获取随机数
  - bac001Address: 积分合约的地址

- start_new_lottery

  由主持人添加所有参与者进来，并开启一个新的抽奖环节，需要传两个参数：

  - _players: 参与者的公钥地址数组
  - _amount: 本轮抽奖需要押注的积分数量

- deposit

  参与者确认自己要参与抽奖，确认后会自动转数额为amount的积分到合约账户下

- stop_deposit

  由主持人结束本轮抽奖的押注，结束后未押注的用户，不能再押注

- pickWinner

  确定胜出者，并将奖池（合约账户）下的所有积分转给胜出者

5.1.3 开发教程

前提

- 已部署randomOracle合约，此合约提供随机数功能;
- 积分合约已部署，并记录地址;
- 相关用户拥有足够的积分，并且允许本合约从自己的账户下转走部分积分;

参与成员说明

- 主持人：weiwei
- 参与抽奖成员：alice、bob
- 提供随机数服务的randomOracle合约
- 用于抽奖质押的 BAC001 合约
- 提供抽奖服务的合约LotteryBacOracle

预准备

- 搭建预言机服务或者从服务供应商获取，得到OracleCore合约地址，如：0xa60a49b75ee98a19939a786d57cd2a9802e4984b
- 用OracleCore合约地址作为初始化参数，部署预言机客户端合约APISampleOracle，记录该客户端合约的地址，如：0x89ad9c94646182534482c7f1277c61aed8c22883
- 部署BAC001合约，给参与抽奖的人员转入足够的积分,并记录该合约的地址，如：0x470e30fbd66fdc02e0dbf08e0cdd9f6cdb068c8a
- 主持人部署抽奖合约LotteryBacOracle,参数是以上得到的APISampleOracle和积分合约地址

抽奖步骤

- 主持人调用抽奖合约LotteryBacOracle的函数start_new_lottery。
  - _players：本轮抽奖入所有参与者
  - amount： 本轮抽奖需要质押的bac001数量
- 所有参与者分别在积分合约中调用approve函数,允许抽奖合约LotteryBacOracle从自己账户下转走部分积分。
- 参与者调用抽奖合约LotteryBacOracle的deposit函数确定参与抽奖，这一步操作后，参与者账户下的积分将会自动转到抽奖合约地址中，额度为本轮抽奖指定的数量。

- 主持人调用抽奖合约LotteryBacOracle的stop_deposit函数停止本轮抽奖，参与者不能继续质押了。
- 主持人调用抽奖合约LotteryBacOracle的pickWinner函数，根据随机数计算出本轮获胜者，并将所有质押的积分转给胜出者。

5.2 区块链盲盒

本项目结合 Truora VRF, 实现一个区块链盲盒功能，并支持盲盒互换。用户可以参考合约逻辑并做相应修改。

5.2.1 业务流程

此业务是一个猫系列的区块链盲盒实现，有7中类型猫可供抽选，类型有 "American Shorthair", "British Shorthair", "Japanese Bobtail", "Chinese Orange", "Russian Blue", "Persian", "Ragdoll"；每一个猫盲盒都是唯一的，用户提供幸运数字（随机数种子）抽取从猫系列中抽取自己的猫盲盒。

用户可以选择将所拥有的猫盲盒进行出售换取对应积分，支持定价出售和拍卖出售。

5.2.2 开发教程

参与成员说明

- Tom：部署合约CatBlindbox、AuctionUnfixedPrice、BAC001
- David：抽取从盲盒抽奖合约抽得一个猫，并在非定价拍卖合约中拍卖这个猫
- Alice、Bob：参与拍卖的成员
- RandomNumberSampleVRF：提供函数获取 vrf 随机数
- BAC001：积分合约
- CatBlindbox:提供盲盒猫抽奖的合约
- AuctionUnfixedPrice：该合约提供拍卖功能

预准备

- 搭建预言机服务或者从服务供应商获取，得到Truora服务方的 VRFCore合约地址和公钥哈希值
- 以VRFCore合约地址和公钥哈希值为构造函数参数，部署用于产生随机数的合约RandomNumberSampleVRF，并记录合约地址
- BAC001合约已部署，并记录地址;
- 相关用户（Alice、Bob）拥有足够的积分，并且允许本合约从自己的账户下转走部分积分;
- 部署非定价拍卖合约AuctionUnfixedPrice
- 部署盲盒猫抽奖合约CatBlindbox

详细步骤

步骤一：盲盒抽奖得到一只猫

1. 由Tom部署盲盒猫抽奖合约CatBlindbox

2. David调用该合约的函数`requestNewBlindboxCat`，并得到该函数返回的 `requestId`
3. David传入`requestId`调用该合约的函数`generateBlindBoxCat`，得到一个随机生成的猫
4. David调用函数`getCatInfo`,查询历次生成的猫信息

步骤二：在非定价拍卖平台发布拍卖消息

1. 由Tom部署非定价拍卖的合约`AuctionUnfixedPrice`
2. David调用该合约的创建函数发布一笔拍卖信息

步骤三：客户参与拍卖这个猫

1. 由Tom部署积分合约
2. Tom分别给客户（Alice、Bob）转一定数量的积分
3. 所有参与拍卖的客户，需要调积分合约的`approve`函数，允许非定价拍卖合约从自己的账户下转走一定数量的积分
   - spender: 允许这个地址从本人账户下转走积分，此处填`AuctionUnfixedPrice`合约地址
   - value: 允许目标地址从自己账下转走积分的数量
4. 客户（Alice、Bob）依次调非定价拍卖合约`AuctionUnfixedPrice`的函数`bid`参与拍卖竞价
5. 拍卖活动到期后，商家（David）调非定价拍卖合约`AuctionUnfixedPrice`的函数`executeSale`结束本轮竞价，本轮出价最高的用户将与商家达成交易。

CHAPTER 6

Truora 子系统

6.1 Truora-Service

6.1.1 概要介绍

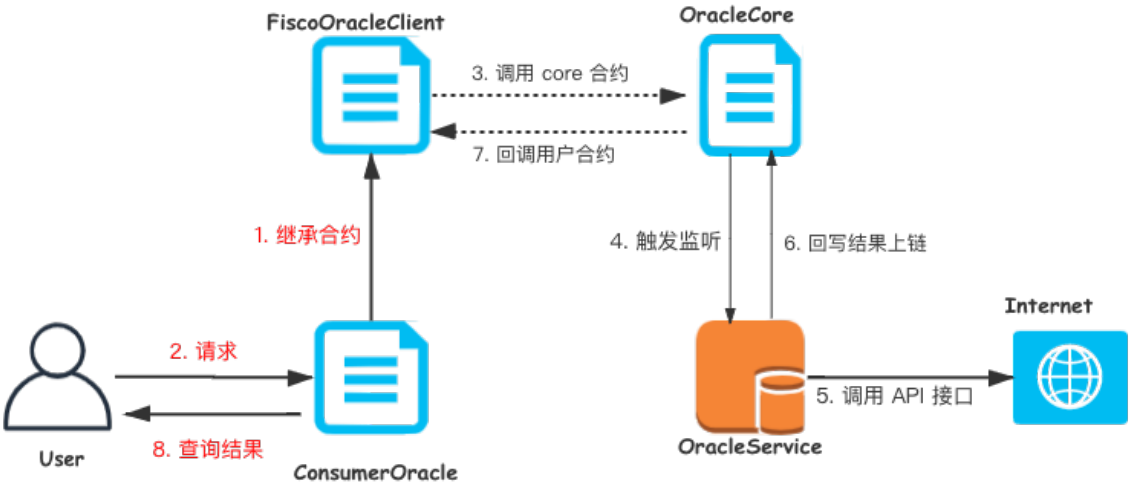
简介

Truora-Service 是 Truora 的后台服务，主要分为链上部分和链下部分。 链上部分主要是 *oracle* 相关合约，链下部分主要是 *Java* 服务，负责连接节点，监听合约的事件，采集结果并回写到 *oracle* 合约。

Truora-Service 支持多集群部署(监听同一条链并共用同一个数据库)。

Truora-Service 目前主要支持获取链下API,链上可验证随机数（VRF），后续会陆续开源去中心化预言机功能。

获取链下API原理图:

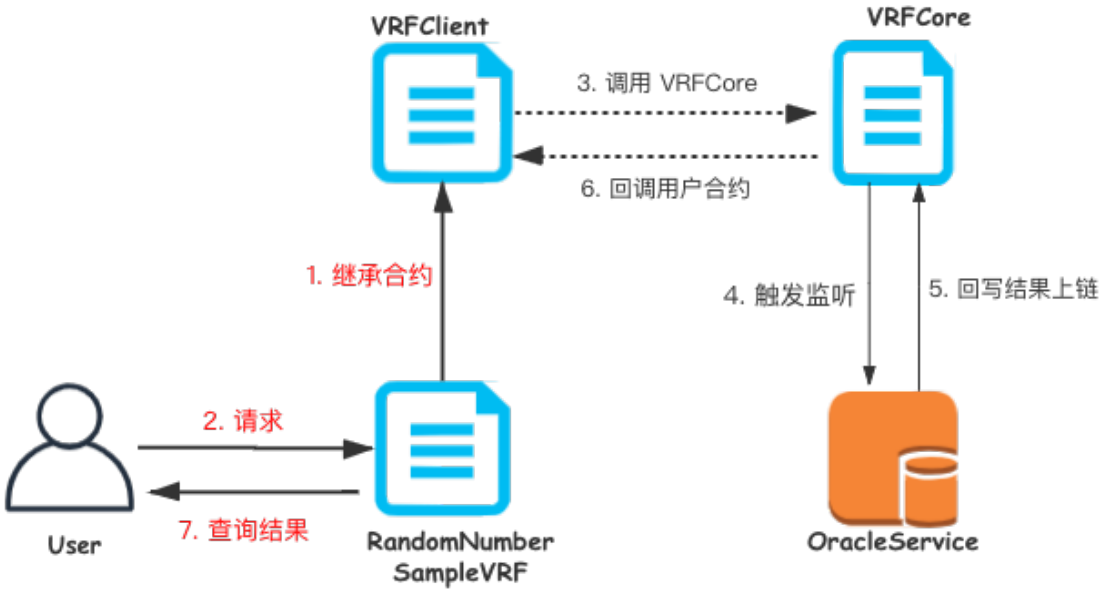


用户发起调用后，FiscoOracleClient 会调用 OracleCore 合约，此时会发起一个事件。Truora-Service 监听到事件后，会从事件信息中取出以下几个信息字段：

- 请求编号（唯一）
- 请求地址和数据解析格式（URL）
- 倍数（防止小数,solidity 不支持浮点数）
- 返回值类型（支持 string，int256, bytes三种）

Truora-Service 获取到数据后，对 URL 发起一个 HTTP 调用，获取到数据，对数据按照解析格式进行处理，然后再调用 OracleCore 合约，根据请求编号将结果上传到链上，提供给其他合约来获取。

获取链上可验证随机数（VRF):



可验证随机函数(Verifiable Random Function ，简写 VRF)是一种将输入映射为可验证的伪随机输出的加密方案。广泛应用于区块链的共识算法，智能合约产生随机数场景中。本方案参照 VRF规范标准化文档实现, VRF原理可以参考文档VRF原理用户发起调用后，需要传入随机数种子，VRFClient 会调用 VRFCore 合约，此时会发起一个事件。Truora-Service 监听到事件后,会从事件信息中取出以下几个信息字段：

- 请求编号（唯一）
- 用户输入随机数种子
- 预言机公钥哈希
- 实际参与VRF计算的随机数种子
- 块高

Truora-Service 获取到数据后，使用服务私钥和实际随机数种子调用vrf函数生成随机数和proof，然后再调用 VRFCore 合约，VRFCore合约会验证proof的准确性，通过后，则根据请求编号将随机数结果回写到用户合约。

提示

- 随机数种子要保持足够随机性，为了安全，实际参与VRF运算的随机数种子是由用户随机数种子，用户合约地址，预言机公钥哈希，用户请求次数，以及当前区块哈希值五个变量哈希处理后的值做为随机数种子。
- 本方案采用的是SECP256K1_SHA256_TAI加密套件，参考实现 ECVRF。
- vrf底层库采用rust实现，proof验证采用solidity实现。
- vrf标准规范中尚未有基于SM2，SM3的实现，本方案暂不支持国密。

6.1.2 源码编译和部署

兼容性说明

- v2stable分支版本所述的truora支持FISCO-BCOS 2.6.0 +
  - Truora-Service 分支v2stable
  - Truora `v1.X.0` 版本支持 FISCO-BCOS 2.6.0 + 版本。
  - 一键安装，Truora docker和web相关模块参见文档说明。
- latest/v3dev分支所述的truora支持FISCO-BCOS 3.1.x +
  - 支持FISCO-BCOS 3.1及以上
  - 文档连接Latest/v3dev分支
  - Truora-Service 主分支main
  - Truora-Service 开发分支v3dev
  - v3dev分支暂不支持一键安装、docker和web模块，欢迎参与开发贡献。

安装介绍

前置依赖

在使用本组件前，请确认系统环境已安装相关依赖软件，清单如下：

请参考：附录 检查系统是否已经安装相关依赖软件。

重要:

- CentOS 的 yum 仓库的 OpenJDK 缺少 JCE(Java Cryptography Extension)，导致 Web3SDK 无法正常连接区块链节点。CentOS 用户推荐参考 CentOS 安装 Java 进行安装。

拉取代码

执行命令：

```
# 拉取源码
git clone https://github.com/WeBankBlockchain/Truora-Service.git

# 进入目录
cd Truora-Service
```

编译代码

方式一：如果服务器已安装Gradle，且版本为 Gradle-4.10 +

```
gradle build -x test
```

方式二：如果服务器未安装 Gradle，或者版本低于 Gradle-4.10，使用 gradlew 编译

```
chmod +x ./gradlew && ./gradlew build -x test
```

构建完成后，会在根目录 Truora-Service 下生成已编译的代码目录 dist。

加密类型

FISCO-BCOS 链有两种类型： **非国密（ECDSA）** 和 **国密（SM2）** 。

在使用 SDK 连接 FISCO-BCOS 链的节点时，也有两种方式：**非国密（ECDSA）连接** 和 **国密（SM2）连接**。

关于 **链类型** 和 **链连接** 的关系如下：

提示

- **非国密** 链 **只支持** 非国密连接
- **国密** 链 **支持** 非国密连接 和 国密连接，但是需要根据节点的 *config.ini* 文件，检查节点是否已经开启国密连接

在部署 Truora-Service，需要同时配置 **链类型** 和 **连接类型**。

链类型

在使用 `build_chain.sh` 脚本部署 FISCO-BCOS 链时，如果使用了 `-g` 参数，则链类型为 **国密链**。

连接类型

如果 FISCO-BCOS 版本小于或等于 `v2.4.x`，**只能使用 非国密方式（ECDSA）** 连接链。

如果 FISCO-BCOS 版本大于或等于 `v2.5.x`，执行命令：

```
# 查看 sm_crypto_channel 配置项
grep "sm_crypto_channel" nodes/127.0.0.1/node0/config.ini
```

- 如果输出：

```
    sm_crypto_channel=true
```

表示节点已经启用 国密（SM2）连接，**只能使用 国密方式（SM2）** 连接链。

- 如果没有输出，或者输出如下：

```
    sm_crypto_channel=false
```

表示节点未启用 国密连接，**只能使用 非国密方式（ECDSA）** 连接链。

重要:

- 不同的连接方式，需要拷贝的证书不同

- 使用 **非国密** 的方式连接节点，需要拷贝 *sdk/* 目录下 *ca.crt*、*node.crt* 和 *node.key* 文件
- 使用 **国密** 的方式连接节点，需要拷贝 *sdk/gm* 目录下 *gm* 开头的所有文件

修改配置

进入 dist 目录

```
cd dist
```

dist 目录提供了一份配置模板 conf

配置数据库

修改配置 conf/application.yml 文件

```
# 进入 conf 目录
cd conf
```

- 修改数据库 IP 地址，用户名和密码。

```
 datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/truora?serverTimezone=GMT%2B8&useUnicode=true&
↪characterEncoding=utf-8&zeroDateTimeBehavior=convertToNull&useSSL=false
    username: "defaultAccount"
    password: "defaultPassword"
```

配置链类型

如果链类型是国密，配置 encryptType： 1，关于链类型，请参考：链类型

```
sdk:
  #0:standard, 1:guomi
  encryptType: 1
```

拷贝证书

在拷贝证书文件之前，需要确定使用哪种方式连接到链接点：非国密连接（ECDSA）还是 国密连接（SM2），请参考：连接类型

- 非国密连接

```
# 进入 conf 目录
cd conf

# 非国密连接
cp  /${PATH_TO_SDK}/node.* .
cp  /${PATH_TO_SDK}/ca.crt .
```

重要:

- **非国密链**： 拷贝节点所在目录 *nodes/${ip}/sdk* 下的 *ca.crt*、*node.crt* 和 *node.key* 文件拷贝到 *conf* 目录

- 国密连接

```
# 进入 conf 目录
cd conf

# 国密连接
cp  /${PATH_TO_SDK}/gm/gm.* .
```

重要:

- **国密链**：拷贝节点所在目录 *nodes/${ip}/sdk/gm* 下的 *gm* 开头的所有文件拷贝到 *conf* 目录

多链（群组）支持

配置连接

Truora-Service 支持同时连接多条链，以及连接同一条链中的多个群组。

同一个 Truora-Service 连接多条链时，要求 **链类型** 都相同，同时采用 **相同的连接方式** 连接到链接点。

重要:

- 不同链之间相互独立，没有关联
- 多条链时，创建独立目录存放不同链的证书文件，同时拷贝证书文件

- 如果采用 **非国密连接（ECDSA）**，修改 `application-ecdsa.yml` 文件

```
########################################################################
# 配置 Truora 连接的链和群组信息（证书和地址）:
#   1. 同一条链可以配置多个群组
#   2. 可以配置多条链
########################################################################
group-channel-connections-configs:
  configs:
    ## 第一条链的连接信息，证书，群组列表和 IP:Port
    - chainId: 1
      caCert: classpath:ca.crt
      sslCert: classpath:node.crt
      sslKey: classpath:node.key
      all-channel-connections:
         - group-id: 1
           connections-str:
             # node listen_ip:channel_listen
             - 127.0.0.1:${FISCO_BCOS_PORT:20200}
        ## 群组 2 的信息
        #- group-id: 2
        #  connections-str:
        #    - 127.0.0.1:20200

    ## 第二条链的连接信息，证书，群组列表以及对应的 IP:Port
    #- chainId: 2
    #  caCert: classpath:2/ca.crt
    #  sslCert: classpath:2/node.crt
    #  sslKey: classpath:2/node.key
    #  all-channel-connections:
    #    - group-id: 1
    #      connections-str:
    #        - 127.0.0.1:20200
```

- 如果采用 **国密连接（SM2）**，修改 `application-sm2.yml` 文件

```
########################################################################
# 配置 Truora 连接的链和群组信息（证书和地址）:
#   1. 同一条链可以配置多个群组
#   2. 可以配置多条链
########################################################################
group-channel-connections-configs:
  configs:
    ## 第一条链的连接信息，证书，群组列表和 IP:Port
    - chainId: 1
      gmCaCert: classpath:gmca.crt
      gmSslCert: classpath:gmsdk.crt
      gmSslKey: classpath:gmsdk.key
      gmEnSslCert: classpath:gmensdk.crt
      gmEnSslKey: classpath:gmensdk.key
      all-channel-connections:
         - group-id: 1
           connections-str:
             # node listen_ip:channel_listen
             - 127.0.0.1:${FISCO_BCOS_PORT:20200}
        ## 群组 2 的信息
        #- group-id: 2
        #  connections-str:
        #    - 127.0.0.1:20200

    ## 第二条链的连接信息，证书，群组列表以及对应的 IP:Port
    #- chainId: 2
    #  gmCaCert: classpath:2/gmca.crt
    #  gmSslCert: classpath:2/gmsdk.crt
    #  gmSslKey: classpath:2/gmsdk.key
    #  gmEnSslCert: classpath:2/gmensdk.crt
    #  gmEnSslKey: classpath:2/gmensdk.key
    #  all-channel-connections:
    #    - group-id: 1
    #      connections-str:
    #        - 127.0.0.1:20200
```

启用链和群组

根据连接类型，修改 `application-ecdsa.yml` 或 `application-sm2.yml` 文件中 `event.eventRegisters` 配置。

提示

- 配置多链多群组监听时，配置的链ID和群组ID，必须在 *group-channel-connections-configs* 中配置过
- *group-channel-connections-configs* 表示 Truora-Service 会连接到哪些链和群组
- *eventRegisters* 表示启用哪些链和群组

```
########################################################################
# 配置事件监听:
#   1. 配置 Truora 需要监听的链（ChainId）和群组（groupId）
#   2. 配置的 chainId 和 groupId 需要在 group-channel-connections-configs 存在
########################################################################
event:
  eventRegisters:
   - {chainId: 1, group: 1}
   #- {chainId: 1, group: 2}
```

(下页继续)

(续上页)

```
    #- {chainId: 2, group: 1}
    #- {chainId: 2, group: 2}
```

服务启停

返回到 dist 目录执行：

- 启动

```
# 采用非国密连接（ECDSA）
bash start.sh

# 采用国密连接（SM2），添加 gm 参数
bash start.sh gm
```

- 停止

```
bash stop.sh
```

- 检查

```
bash status.sh
```

备注：服务进程起来后，需通过日志确认是否正常启动，出现以下内容表示正常；如果服务出现异常，确认修改配置后，重启提示服务进程在运行，则先执行 stop.sh，再执行 start.sh。

```
Application() - main run success...
```

查看日志

在 dist 目录查看：

```
# 前置服务日志:
tail -f log/Oracle-Service.log
```

6.1.3 接口规范

Truora-Service

查询 Truora 版本

接口描述

查询 Truora 服务版本

接口URL

http://localhost:5021/Oracle-Service/server/version

调用方法

HTTP GET

请求参数

1）参数表

无

2）数据格式

无

响应参数

1）数据格式

```
{
  "code": 0,
  "message": "success",
  // 当前 Truora-Service 版本
  "data": "v1.1.0",
  "totalCount": 0
}
```

OracleCore 合约地址查询接口

接口描述

查询 OracleCore 合约地址，用户编写自定义合约时使用

接口URL

http://localhost:5021/Oracle-Service/oracle/address?chainId=1&groupId=1

调用方法

HTTP GET

请求参数

1）参数表

2）数据格式 无

响应参数

1）数据格式

```
{
  "code": 0,
  "message": "success",
  "data": [
    {
      // OracleCore 合约版本
      "oracleCoreVersion": "v1.1.0",
```

(下页继续)

(续上页)

```
      // VRFCore 合约版本
      "vrfCoreVersion": "v1.1.0",
      // 链 ID
      "chainId": 1,
      // 群组 ID
      "group": 1,
      // OracleCore 合约地址
      "oracleCoreContractAddress": "0x1769c87d9889eee2352c9cfe61dd3ac22a80f7a7",
      // VRFCore 合约地址
      "vrfContractAddress": "0x28b2a1b9ead2e4cf6a7cff9372ee6da70152e484",
      "fromBlock": "latest",
      "toBlock": "latest"
    }
    .....
  ],
  "totalCount": 0
}
```

查询历史请求列表

接口描述

查询所有的历史请求记录

接口URL

http://localhost:5021/Oracle-Service/history/list?pageNumber=1&pageSize=10&chainId=1&groupId=1&hideResult=false

调用方法

HTTP GET

请求参数

1）参数表

2）数据格式 无

响应参数

1）数据格式

```
{
  "code": 0,
  "message": "success",
  "data": [
    // 多条记录
    {// sourceType == 0 时，表示链下 API 获取数据
      "id": 1,
      //请求唯一编号
      //查询详情时使用的 requestId 字段
      "reqId":
"0x9f32a5e56608fd730f7ef8bc42efdc53142753a77e46287246ed2a9a39ed1994",
```

(下页继续)

(续上页)

```
      // 链编号和群组编号
      "chainId": 1,
      "groupId": 1,

      // Core 合约的版本号
      "oracleVersion": "v1.1.0",

      // 请求类型
      //  0 API 获取链下数据
      "sourceType": 0,

      // URL 请求求地址和数据响应格式
      "reqQuery": "json(https://api.exchangerate-api.com/v4/latest/CNY).rates.JPY",

      // 请求状态
      // 0 成功
      // 非 0，失败
      "reqStatus": 0,
      // 请求状态非 0 时的错误信息
      "error": "",

      // 发起调用的合约地址
      "userContract": "0x919c4e3c50a074dbd15e6a832bc146cd288cebf4",

      // 请求耗时（ms）
      "processTime": 2460,

      // hideResult 为 true，不返回该字段
      // hideResult 为 false，返回
      // URL 返回的结果
      "result": "15.962067",

      // 放大倍数，Truora-Service 会将 URL 返回的结果 乘以 该倍数后上传到链上
      // 防止小数
      "timesAmount": "1000000000000000000",

      "createTime": "2020-12-11 18:48:28",
      "modifyTime": "2020-12-11 18:48:31"
    },
    {// sourceType == 1 时，表示 VRF，获取链上随机数
      "id": 53,
      //请求唯一编号
      //查询详情时使用的 requestId 字段
      "reqId": "0x223c4a975b25b235b0f3e9830e1099cdfefeb7dcffa839b860fccd12a6cd0a51
↪",
       // 链编号和群组编号
      "chainId": 1,
      "groupId": 1,

      // Core 合约的版本号
      "oracleVersion": "v1.1.0",

      //请求类型
      // 1 VRF 获取可验证随机数
      "sourceType": 1,
      "blockNumber": 146,
      "reqQuery": "",
      "needProof": false,

      // 请求状态
```

(下页继续)

(续上页)

```
      // 0 成功
      // 非 0，失败
      "reqStatus": 0,
      // 请求状态非 0 时的错误信息
      "error": "",

      // 发起调用的合约地址
      "userContract": "0xdd48ff8868b76dab43bd13402840bcb8c5a82853",

      // 处理时长
      "processTime": 495,

      "createTime": "2021-03-22 16:28:56",
      "modifyTime": "2021-03-22 16:28:56",
      "inputSeed": "1",
      "actualSeed":
↪"BF18046A6A8801B4ABED36169F93D1C0613768EC0EDCC827A31D973D3CBE98B4"
    },
    ......
  ],
  // 总行数
  "totalCount": 1
}
```

查询单个请求详情

接口描述

查询单个请求详情

接口URL

http://localhost:5021/Oracle-Service/history/query/{requestId}

调用方法

HTTP GET

请求参数

1）参数表

注意：**requestId** 对应历史请求列表中返回记录的 **reqId** 字段

2）数据格式 无

响应参数

1）数据格式

```
{
  "code": 0,
  "message": "success",
```

(下页继续)

(续上页)

```
  "data": {
    "id": 1,
    //请求唯一编号
    //查询详情时使用的 requestId 字段
    "reqId": "0x9f32a5e56608fd730f7ef8bc42efdc53142753a77e46287246ed2a9a39ed1994",

    // 链编号和群组编号
    "chainId": 1,
    "groupId": 1,

    // Core 合约的版本号
    "oracleVersion": "v1.1.0",

    // 请求类型
    // 0 链下 API 获取数据
    // 1 VRF 获取可验证随机数
    "sourceType": 1,

    // sourceType == 0 时，表示：URL 请求地址和数据响应格式
    // sourceType == 1 时，不使用
    "reqQuery": "json(https://api.exchangerate-api.com/v4/latest/CNY).rates.JPY",

    // 请求状态
    // 0 成功
    // 非 0，失败
    "reqStatus": 0,
    // 请求状态非 0 时的错误信息
    "error": "",

    // 发起调用的合约地址
    "userContract": "0x919c4e3c50a074dbd15e6a832bc146cd288cebf4",

    // 请求耗时（ms）
    "processTime": 2460,

    // hideResult 为 true，不返回该字段
    // hideResult 为 false，返回
    // sourceType == 0 时，表示：URL 返回的结果
    // sourceType == 1 时，表示：VRF 生成的随机数，0x 开头 16 进制
    "result": "15.962067",

    // 放大倍数，防止小数
    // sourceType == 0 时使用，Truora-Service 会将 URL 返回的结果 乘以 该倍数后上传到链上
    "timesAmount": "1000000000000000000",

    // sourceType == 1 时使用，VRF 生成的可验证随机数的证明
    "proof":
↪"0211c851316725ffba00e7eae4a06327214e282803379a7521ade5bbd0dc3a4ba5c69f06f471fd2349e907ca28df48
↪"
    // sourceType == 1 时使用，用户提供的随机数种子
    "inputSeed": "1",
    // sourceType == 1 时使用，生成随机数时使用的实际种子（包含块高 hash）
    "actualSeed": "1298E2D458FA77C078D42B4B3B54441D56E4A13ECDE436E31A4899BB0D1A3D74
↪"

    // 额外字段
    "createTime": "2020-12-11 18:48:28",
    "modifyTime": "2020-12-11 18:48:31"
  },
  "totalCount": 0
}
```

查询 Truora-Service 服务信息

接口描述

查询 Truora-Service 服务信息，包括 `keyHash` 和 `publicKeyList`

接口URL

http://localhost:5021/Oracle-Service/center/list?chainId=1&groupId=1

调用方法

HTTP GET

请求参数

1）参数表

2）数据格式 无

响应参数

1）数据格式

```
{
  "code": 0,
  "message": "success",
  "data": [
    {
      // 服务编号，从 0 开始，一次递增
      "index": 0,
      //
      "oracleServiceAddress": "0x9c9c89314573086ace5a5825b33d52eee1f99a8a",
      // Truora-Service 的 PublicKey
      "publicKeyList": [
        "1c8f2ab90b4323f182e85fcd25e4d8b17267b9decb1305592b3d66952ce3d82a",
        "008e89fdc1b5807c400e6339eb5428318be0d5a09696693ce40f27eede2d162a56"
      ],
      // Truora-Service 的 keyHash
      "keyHash": "45f6483e01a8956d4ce4700d9c9c89314573086ace5a5825b33d52eee1f99a8a
↪",
      "operator": "operator",
      "url": "http://localhost",
      "creatTime": "2020-11-18 11:40:12",
      "latestRequstProcessedTime": 0,
      "status": true,
      "processedRequestAmount": 0
    },
    // 多个
    ......
  ],
  "totalCount": 0
}
```

6.1.4 常见问题

6.1.5 附录

安装 Docker

环境要求

安装 Docker

Docker 官方提供了一键安装工具，可以方便的在主机上安装 Docker 工具。

选择一个镜像云安装 Docker 服务：

```
# （推荐）使用 阿里云镜像 安装 Docker
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun;

# 使用 微软云镜像 安装 Docker
curl -fsSL https://get.docker.com | bash -s docker --mirror AzureChinaCloud;
```

如果安装出现下面类似，提示 containerd.io 版本错误：

```
Last metadata expiration check: 0:13:10 ago on Sun 08 Mar 2020 04:23:54 AM UTC.
Error:
 Problem: package docker-ce-3:19.03.7-3.el7.x86_64 requires containerd.io >= 1.2.2-
↪3, but none of the providers can be installed
  - cannot install the best candidate for the job
  - package containerd.io-1.2.10-3.2.el7.x86_64 is excluded
  - package containerd.io-1.2.13-3.1.el7.x86_64 is excluded
  - package containerd.io-1.2.2-3.3.el7.x86_64 is excluded
  - package containerd.io-1.2.2-3.el7.x86_64 is excluded
  - package containerd.io-1.2.4-3.1.el7.x86_64 is excluded
  - package containerd.io-1.2.5-3.1.el7.x86_64 is excluded
  - package containerd.io-1.2.6-3.3.el7.x86_64 is excluded
(try to add '--skip-broken' to skip uninstallable packages or '--nobest' to use␣
↪not only best candidate packages)
```

需要执行一下下面的命令，然后再重新执行安装命令：

```
# 安装 containerd.io 文件
yum -y install "https://download.docker.com/linux/centos/7/x86_64/stable/Packages/
↪containerd.io-1.3.9-3.1.el7.x86_64.rpm"

# 重新执行 Docker 安装命令
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun;
```

- 启动 Docker

```
# 启动 Docker
systemctl start docker
```

- 检测 Docker 安装

```
# 启动 Hello World 容器
$ docker run --rm hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
```

(下页继续)

(续上页)

```
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

如果出现以上输出，包含 `Hello from Docker!`，表示 Docker 安装成功。

安装 Docker-Compose

Docker-Compose 是 Docker 官方提供的基于单机的容器编排工具，可以很方便的在单台主机中管理多个容器，包括按照依赖顺序启动，关闭，重启等。

- 选择一种方式安装 Docker Compose：

```
# （推荐）从 daocloud 下载 docker-compose
curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-compose-
↪$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose;

# 从 官网下载 docker-compose
curl -L "https://get.daocloud.io/docker/compose/releases/download/1.27.4/docker-
↪compose-`uname -s`-`uname -m`" -o /usr/local/bin/docker-compose;
```

- `docker-compose` 文件添加执行权限

```
# docker-compose 文件添加执行权限
chmod +x /usr/local/bin/docker-compose;
```

- 检测 Docker Compose 安装

```
# 输出 Docker Compose 版本号
$ docker-compose -v
docker-compose version 1.27.4, build 40524192
```

安装 Java

Ubuntu（Debian） 安装 Java

```
# 安装默认Java版本(Java 8或以上)
sudo apt install -y default-jdk
# 查询Java版本
java -version
```

CentOS 安装 Java

```
# 查询 CentOS 原有的 Java 版本
$ rpm -qa|grep java

# 删除查询到的Java版本
$ rpm -e --nodeps java-[VERSION]

# 查询 Java 版本，没有出现版本号则删除完毕
$ java -version

# 创建新的文件夹，安装Java 8或以上的版本，将下载的jdk放在software目录
# 从 openJDK官网 (https://jdk.java.net/java-se-ri/8)
#   或
# Oracle官网(https://www.oracle.com/technetwork/java/javase/downloads/index.html)
# 选择Java 8或以上的版本下载
# 例如下载jdk-8u201-linux-x64.tar.gz
$ mkdir /software

# 解压jdk
$ tar -zxvf jdk-8u201-linux-x64.tar.gz

# 配置Java环境，编辑/etc/profile文件
$ vim /etc/profile

# 打开以后将下面三句输入到文件里面并退出
export JAVA_HOME=/software/jdk-8u201-linux-x64.tar.gz
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar

# 生效profile
$ source /etc/profile
# 查询Java版本，出现的版本是自己下载的版本，则安装成功。
java -version
```

安装 Nginx

安装 Nginx 时，推荐使用 yum/apt-get 安装。

- **CentOS 7.x / RHEL 7.x**

```
# 安装 EPEL 仓库
sudo yum install epel-release

# 更新仓库
sudo yum update

# 安装 Nginx
sudo yum install -y nginx
```

- **Ubuntu 16.04 / Debian 9**

```
# 更新仓库
sudo apt-get update

# 安装 Nginx
sudo apt-get -y install nginx
```

- 测试是否安装成功

查看 Nginx 版本:

```
$ nginx -v

nginx version: nginx/1.16.1
```

查看 Nginx 配置文件路径:

```
$ nginx -t

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

安装 MySQL

此处以 CentOS 安装 MariaDB 为例。MariaDB 数据库是 MySQL 的一个分支，主要由开源社区在维护，采用 GPL 授权许可。MariaDB 完全兼容 MySQL，包括 API 和命令行。其他安装方式请参考MySQL官网。

（1）安装 MariaDB

- 安装命令

```
sudo yum install -y mariadb*
```

（2）启停

```
启动: sudo systemctl start mariadb.service
停止: sudo systemctl stop  mariadb.service
```

（3）设置开机启动

```
sudo systemctl enable mariadb.service
```

（4）初始化root用户

```
# 执行以下命令:
sudo mysql_secure_installation

#以下根据提示输入:

Enter current password for root (enter for none):<-初次运行直接回车
Set root password? [Y/n] <- 是否设置root用户密码，输入y并回车或直接回车
New password: <- 设置root用户的密码
Re-enter new password: <- 再输入一次你设置的密码
Remove anonymous users? [Y/n] <- 是否删除匿名用户，回车
Disallow root login remotely? [Y/n] <-是否禁止root远程登录，回车
Remove test database and access to it? [Y/n] <- 是否删除test数据库，回车
Reload privilege tables now? [Y/n] <- 是否重新加载权限表，回车
```

- 使用root用户登录，密码为初始化设置的密码

```
mysql -uroot -p -h localhost -P 3306
```

- 授权root用户远程访问

- 注意，以下语句仅适用于开发环境，不能直接在实际生产中使用！！！ 以下操作仅供参考，请勿直接拷贝，请自定义设置复杂密码。

```
mysql > GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY '123456' WITH␣
↪GRANT OPTION;
mysql > flush PRIVILEGES;
```

提示

- 例子中给出的数据库密码（*123456*）仅为样例，强烈建议设置成复杂密码
- 例子中 *root* 用户的远程授权设置会使数据库在所有网络上都可以访问，请按具体的网络拓扑和权限控制情况，设置网络和权限帐号

安全温馨提示:

（5）创建test用户并授权本地访问

```
mysql > GRANT ALL PRIVILEGES ON *.* TO 'test'@localhost IDENTIFIED BY '123456'␣
↪WITH GRANT OPTION;
mysql > flush PRIVILEGES;
```

（6）测试是否成功

- 登录数据库

```
mysql -utest -p123456 -h localhost -P 3306
```

- 创建数据库

```
mysql > create database datastash;
mysql > use datastash;
```

以上语句仅适用于开发环境，不能直接在实际生产中使用！！！以上设置会使数据库在所有网络上都可以访问，请按具体的网络拓扑和权限控制情况，设置网络和权限帐号

Git安装

Git：用于拉取最新代码

CentOS::

```
sudo yum -y install git
```

Ubuntu:

```
sudo apt install git
```

6.2 Truora-Web

6.2.1 概要介绍

简介

Truora-Web 是 Truora 服务的前端组件。 主要支持请求详情和请求历史的查询。方便应用查询自己预言机请求的结果，如果失败，可以看到请求失败的原因。

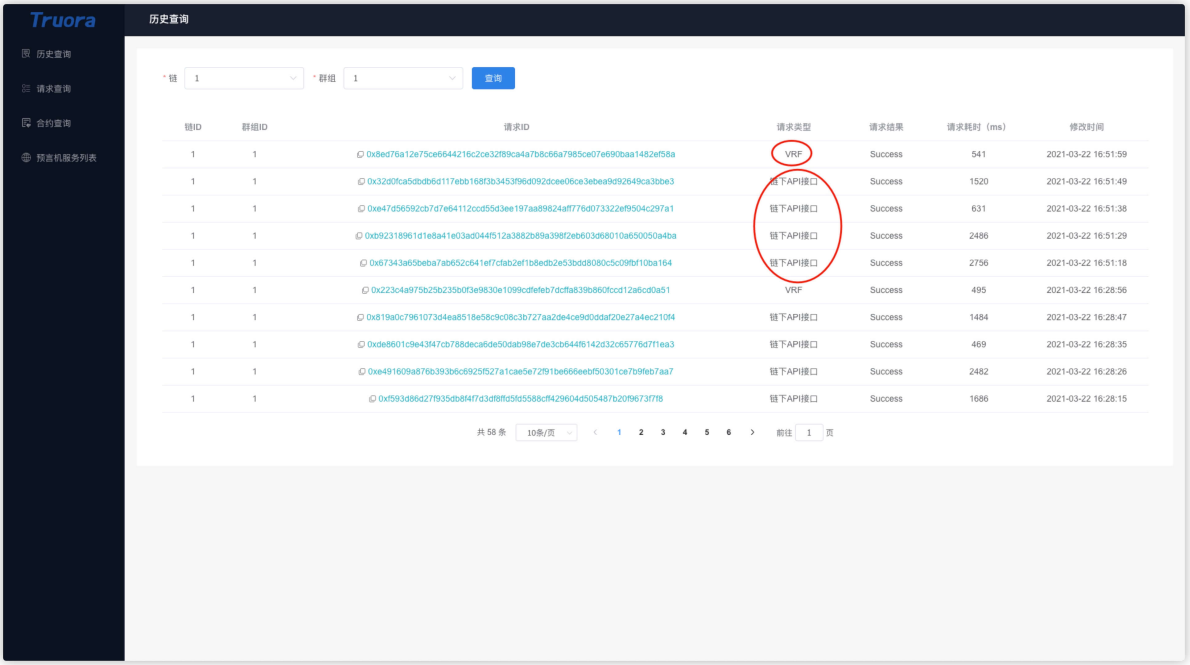
该项目是区块链中间件 **Truora-Service 预言机** 的前端服务，基于 `vue-cli` 框架开发。

兼容浏览器 `IE9` 及以上，`360` 浏览器兼容版（`IE9` 内核），`360` 浏览器极速版，`Chrome` 浏览器。

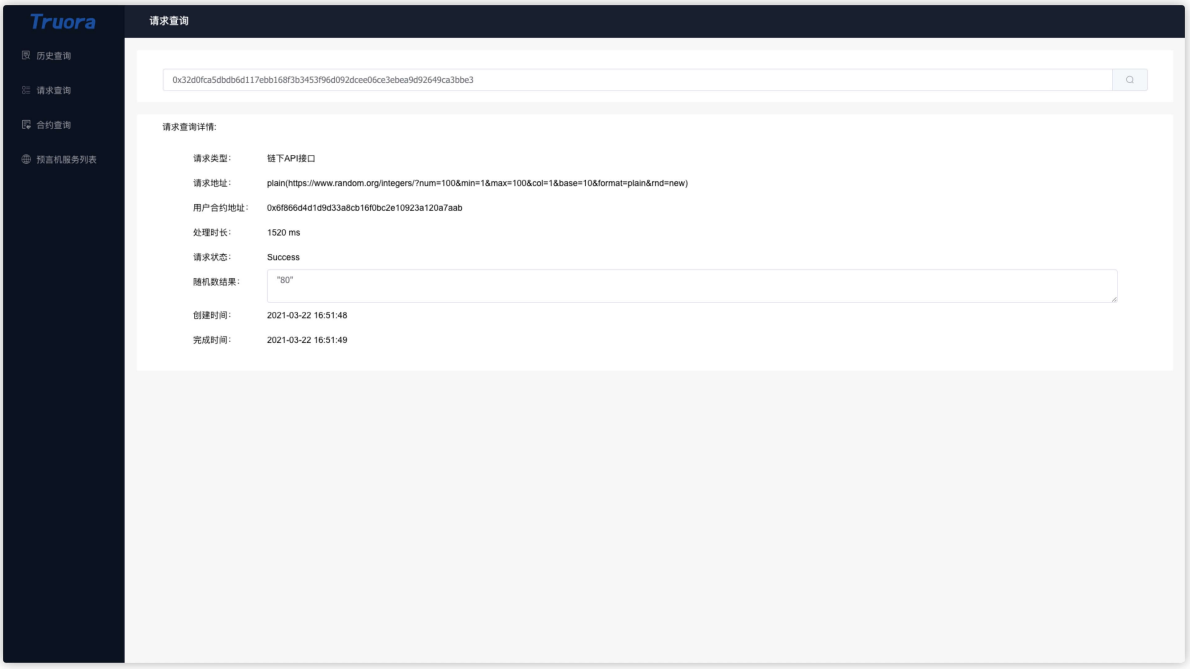
具体功能如下：

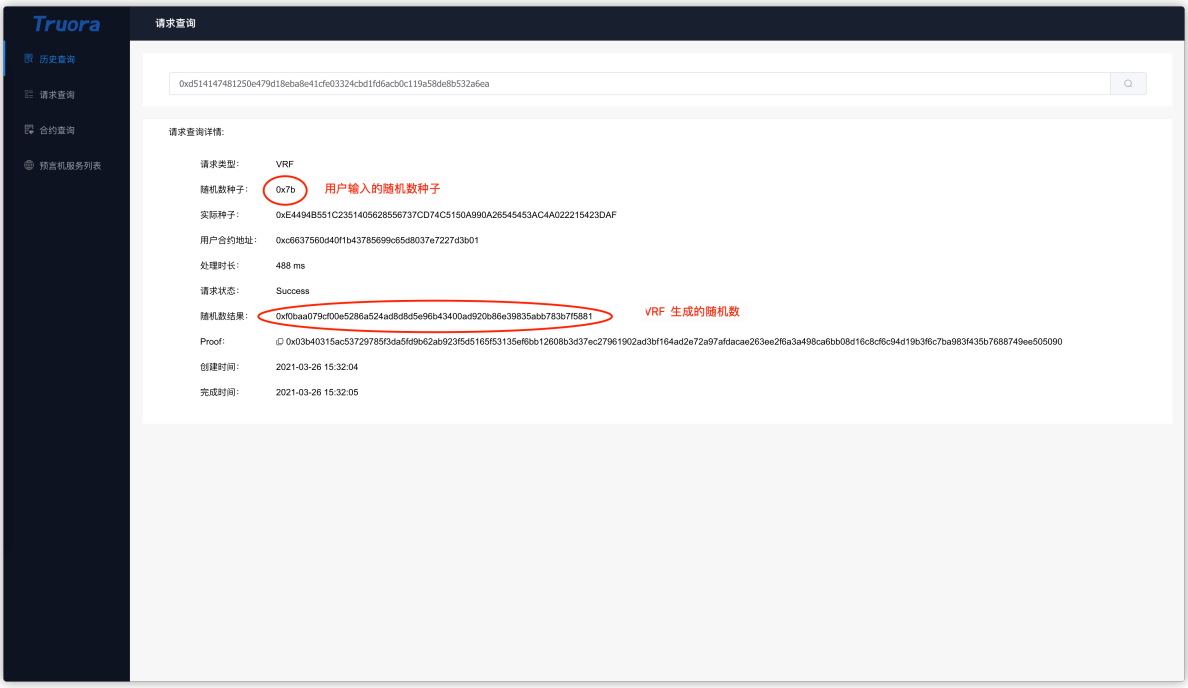
- **分页查询请求记录：**

- 通过 **链 ID** 和 **群组 ID**，分页查询请求历史记录

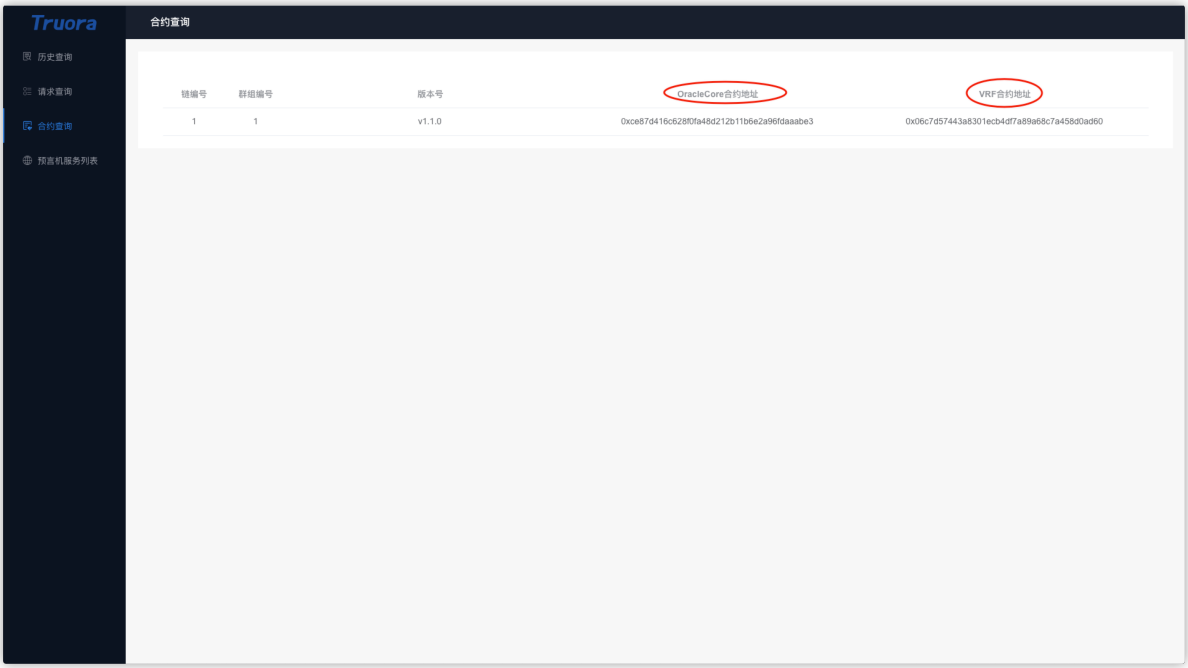


- **查询单个请求详情**：
  - 根据 **请求 ID**，查询单个请求的详细信息：成功，失败（失败原因），处理时长等





- **查询系统合约地址：**
  - 查询 `OracleCore` 合约的地址，编写业务合约时使用



6.2.2 源码部署

依赖环境

关于 `Nginx` 安装，请参考 附录–安装 Nginx

安装

- 拉取代码：

```
# 拉取代码
git clone https://github.com/WeBankBlockchain/Truora-Web.git;

# 进入目录
cd Truora-Web
```

- 拷贝 Nginx 配置文件

```
# 拷贝项目的 Nginx 配置
cp docker/truora-web.conf  /etc/nginx/conf.d
```

提示

- 如果主机中已经存在 Nginx，只需要把 *docker/truora-web.conf* 拷贝到 Nginx 的配置文件（默认：*/etc/nginx/nginx.conf*）中 *include* 的目录下即可。

重要:

- Nginx 的配置文件直接拷贝 *docker/truora-web.conf* 文件即可，**不建议** 修改 Nginx 的主配置文件（默认：*/etc/nginx/nginx.conf*）

- 拷贝源码到 Nginx 项目目录

```
# 备份原有 index.html 文件
mv /usr/share/nginx/html/index.html /usr/share/nginx/html/index.html.back

# 拷贝项目文件
cp -r dist/* /usr/share/nginx/html
```

启动 Nginx

- 启动 Nginx

```
# 启动 Nginx
nginx
```

- 检查启动结果

```
# 检查进程
$ ps -ef | grep -i nginx

root     31462     1  0 17:33 ?        00:00:00 nginx: master process nginx
www-data 31562 31462  0 17:37 ?        00:00:00 nginx: worker process
www-data 31563 31462  0 17:37 ?        00:00:00 nginx: worker process

# 检测端口
$ netstat -anp | grep -i listen| grep -i tcp|grep 5020

tcp        0      0 0.0.0.0:5020            0.0.0.0:*               LISTEN      ␣
↪31462/nginx: master
```

如果能查看到 nginx 的进程，同时 5020 端口也被 Nginx 服务监听，那么 **启动成功**。

- 浏览器访问

  打开浏览器，输入 `http://[IP]:5020`，比如：`http://127.0.0.1:5020`

提示

- 注意替换服务器的 *IP* 地址

错误排查

- 查看 `Nginx` 日志：

```
# 查看 Nginx 的错误日志
$ cat /var/log/nginx/error.log
```

- 查看项目的 `Nginx` 日志：

```
$ 项目的 error 日志
cat /var/log/nginx/oracle-error.log

$ 项目的 access 日志
cat /var/log/nginx/oracle-access.log
```

6.2.3 常见问题

CHAPTER 7

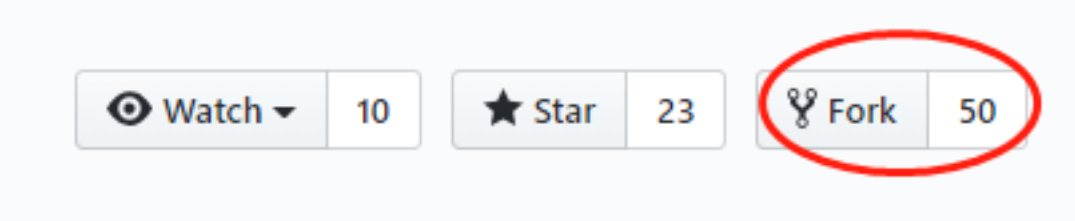
Truora 贡献指南

欢迎，提前感谢你的帮助和支持！

如果你是第一次贡献，只需按照以下简单步骤操作即可。我们将以修改为 Truora-Service 例子给你介绍。

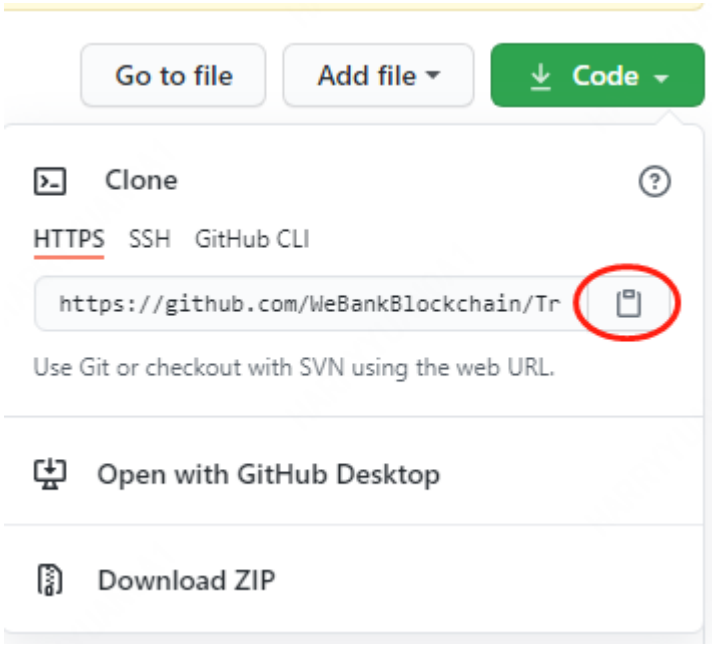
如果你的电脑上尚未安装 `git`, 请按照这个安装指引 进行安装。

7.1 Fork本代码仓库



点击图示中的按钮去 Fork 这个代码仓库。 这个操作会将代码仓库复制到你的账户名下。

7.2 Clone代码仓库



接下来，将复制后的代码仓库克隆到你的电脑上。点击图示中的绿色按钮，接着点击复制到剪切板按钮（将代码仓库地址复制下来）

随后打开命令行窗口，敲入如下 git 命令：

```
git clone https://github.com/{WeBankBlockchain}/Truora-Service.git
```

替换大括号为自己的账号，将 fork 后的仓库 Truora-Service 克隆到你的本地电脑上。

7.3 代码修改

```
cd Truora-Service

vim XXX
```

7.4 Commit修改

```
git commit -m "A great commit"
```

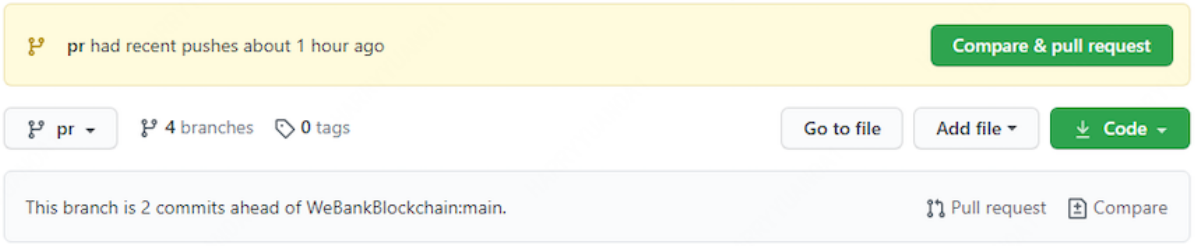
7.5 将改动 Push 到 GitHub

使用 git push 命令发布代码，替换大括号中 dev 为自己的分支。

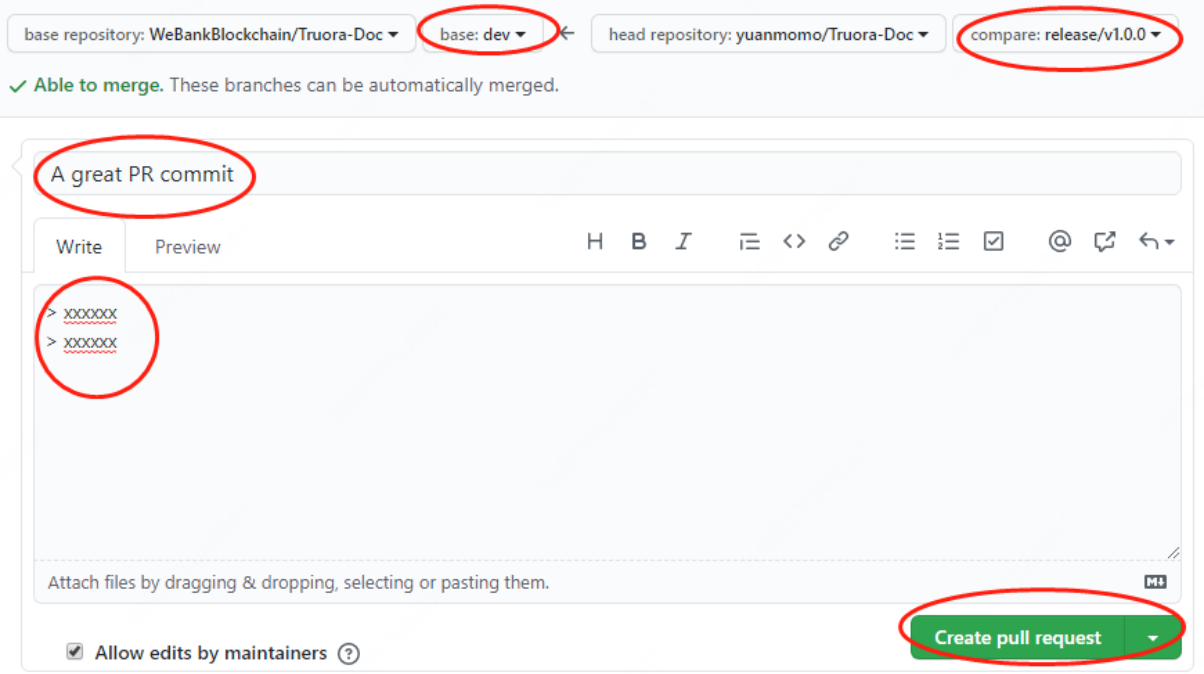
```
git push origin {dev}
```

7.6 提出 Pull Request 将你的修改供他人审阅

前往 Github 你的代码仓库，你会看到一个 `Compare & pull request` 的按钮，点击该按钮。



选择自己的分支，填写 PR 的标题和明细，接着再点击 `Create pull request` 按钮，正式提交 pull request。



提交的改动经过审核，会合入到仓库。合并后，你会收到电子邮件通知。

CHAPTER 8

社区

8.1 加入FISCO BCOS社区

来Meetup畅聊技术

走出去拓展区块链人脉 | 打破技术认知边界

- 全国巡回进行时 -

<< 精彩继续

广州 | 北京 | 上海 | 西安

活动日程安排及报名信息
请关注公众号

利用区块链找靠谱月嫂

成为贡献者

希望以后你可以拿这个项目给自己加分：
“FISCO BCOS是我一手搞起来的！”

★ Star	于你是收藏，于我是鼓励
New issue	反馈bug	问题交流
New PR	文档修改	bug修复	提交新功能特性