技术方案

1 背景和目标

背景

为了熟悉 Rust 开发及 Rust 常用数据结构，本项目采用区块链技术实现一个简易的区块链 Demo，重点在于数据结构设计、哈希计算、安全性及基本交互。

目标

- **实现初始块的创建**
- **能够增加新的数据块**
- **提供基本的交互体验**
- **保证交易安全性**
- **支持区块链数据持久化**

2. 方案概述

该区块链采用 **Rust 语言** 开发，包含以下核心功能：

1. **区块创建**：生成创世区块，并支持新增区块。
2. **交易存储**：每个区块存储交易信息。
3. **工作量证明（PoW）**：计算区块哈希，防止篡改。
4. **区块校验**：验证区块完整性，确保链上数据安全。
5. **数据持久化**：存储区块数据，重启后可恢复。

3. 关键技术

模块	技术选型
编程语言	Rust
存储	JSON（可扩展）
加密算法	SHA3-256（区块哈希）
共识机制	PoW（工作量证明）
数据结构	Vec（链式存储）
交互	CLI 交互（可扩展）

4. 用户故事 & 技术实现

4.1 用户故事 - 创建初始块

用户需求：

- 作为用户，我希望通过运行程序创建初始块。

验收标准：

- 运行程序后，终端打印出创世区块。

技术实现：

- 设计 `Block` 结构体，包含 `header`、`hash`、`data`。
  - 其中 `header` 为块头、`hash` 为块hash、`data` 为区块携带数据
- `header` 的结构体为 `BlockHeader`，包含 `time`、`tx_hash`、`pre_hash`、`difficulty`、`nonce`
  - 其中 `time` 为区块生成的时间戳
  - `tx_hash` 为当前块交易hash，保存教学数据
  - `pre_hash` 为前一个块的hash
  - `difficulty` 为工作量证明的难度系数
  - `nonce` 为工作量证明的随机数
- 创建 `Blockchain` 结构体，包含 `difficulty` 记录当前链工作量证明的难度系数，`blocks` 链存储数组
- 初始化创世区块（Genesis Block）
  - 调用 `new_block_chain` 返回一个链对象
- 运行 `main.rs`，打印链和初始区块信息。

4.2 用户故事 - 增加交易

用户需求：

- 作为用户，我希望区块链能够记录我的每一笔交易。

验收标准：

- 用户可以添加新交易块。
- 可以查询区块链，保证交易未被篡改。

技术实现：

- 调用 `Blockchain::add_block(data: String)` 添加新区块，其中调用 `Block::newblock()` 返回工作量证明后的区块，存入 `Blockchain` 中的 `blocks`。
- 同样运行 `main.rs`，打印链和初始区块信息。

4.3 用户故事 - 交易安全

用户需求：

- 作为用户，我希望生成区块时通过有效的工作量证明生成 `hash`，防止链上数据被恶意更改。

验收标准：

- 实现 PoW 机制，在生成区块时进行哈希验证。
- 校验区块链完整性，防止篡改。

技术实现：

- 在`Block::newblock()`过程中有`Block::proof_of_work()`通过计算前缀为困难系数个数的连续0的hash值为块hash， 实现 PoW。
- 通过`Blockchain::block_chain_is_valid()` 递归校验所有区块的`pre_hash`和`hash`直到创世区块。

4.4 用户故事 - 持久化

用户需求：

- 作为用户，我希望当我打开程序时，能保留之前的交易记录，同时在交易记录被修改时无法存入新的块。

验收标准：

- 启动项目可查看之前的区块数据。
- 当数据被恶意篡改后，无法添加新块。

技术实现：

- 使用 `serde_json` 进行 JSON 序列化，为方便演示hash篡改用明文保存。
- `db::save_chain()` 在产生新区块时进行持久化存储（包括创世区块）。
- `db::load_chain()` 读取本地数据作为当前链。
- `db::is_chain_exist()`判断是否有持久化文件，如果没有就创建新链，有就加载文件。

5. 未来优化方向

优化方向	方案
性能优化	多线程挖矿
存储优化	使用 RocksDB 代替 JSON 文件存储
共识优化	实现 PoS（权益证明）提高效率
交互优化	提供 Web API
安全优化	引入Merkle Root保证所有交易的安全