技术方案

1 技术栈

- **编程语言**：Rust
- **工具库**：
  - `bincode`：用于序列化和反序列化。
  - `serde`：支持 `bincode` 的序列化。
  - `crypto`：用于哈希计算。
  - `chrono`：用于生成时间戳。

2 系统架构

1. **区块结构**：
   - **区块头:**
   - `time`: 时间戳
   - `tx_hash`: data 的哈希
   - `pre_hash`：前一个区块的哈希值
   - `hash`：当前区块的哈希值
   - `data`：区块数据（简单字符串）
2. **区块链结构**：
   - `BlockChain`：区块的链表
   - `add_block`：添加新区块的方法
   - `new_genesis_block`：新建创世区块的方法
   - `new_blockchain`：新建区块链的方法

3 核心逻辑

1. **创世区块生成：**
    - 系统启动时，生成一个创世区块，`pre_hash` 为空，`data` 为“This is genesis block”。
2. **新区块添加：**
    - 通过程序逻辑自动添加新区块，区块数据为简单的字符串（如“This is bolck 1”）。
3. **区块链展示：**
    - 程序运行结束后，遍历区块链，输出每个区块的详细信息。

4 代码结构

```
src/
├── main.rs             # 主程序入口，区块添加逻辑
├── block.rs            # 区块结构定义
├── blockchain.rs       # 区块链结构定义
└── coder.rs            # 工具函数（序列化和反序列化，求哈希）
```

区块结构

```rust
#[derive(Serialize, Deserialize, Debug, PartialEq)]
pub struct BlockHeader {
    pub time: i64,
    pub tx_hash: String,
    pub pre_hash: String,
}

#[derive(Debug)]
pub struct Block {
    pub header: BlockHeader,
    pub hash: String,
    pub data: String,
}
```

区块链结构

```
pub struct BlockChain {
    pub blocks: Vec<block::Block>,
}

impl BlockChain {
    pub   fn add_block(&mut self, data: String) {
        let pre_block = &self.blocks[self.blocks.len()-1];
        let new_block = block::Block::new_block(data, pre_block.hash.clone());
        self.blocks.push(new_block);
    }

    fn new_genesis_block() -> block::Block {
        block::Block::new_block("This is genesis block".to_string(), String::from(""))
    }

    pub fn new_blockchain() -> BlockChain {
        BlockChain {
            blocks: vec![BlockChain::new_genesis_block()],
        }
    }
}
```

程序逻辑示例

```
use chrono::prelude::*;
use utils::coder;
use serde::{Deserialize, Serialize};

#[derive(Serialize, Deserialize, Debug, PartialEq)]
pub struct BlockHeader {
    pub time: i64,
    pub tx_hash: String,
    pub pre_hash: String,
}

#[derive(Debug)]
pub struct Block {
    pub header: BlockHeader,
    pub hash: String,
    pub data: String,
```

```
}

pub struct BlockChain {
    pub blocks: Vec<block::Block>,
}

impl BlockChain {
    pub   fn add_block(&mut self, data: String) {
        let pre_block = &self.blocks[self.blocks.len()-1];
        let new_block = block::Block::new_block(data, pre_block.hash.clone());
        self.blocks.push(new_block);
   }

    fn new_genesis_block() -> block::Block {
        block::Block::new_block("This is genesis block".to_string(), String::from(""))
   }

    pub fn new_blockchain() -> BlockChain {
        BlockChain {
            blocks: vec![BlockChain::new_genesis_block()],
        }
   }
}
```
