技术方案

1. 技术选型

- **编程语言**：Rust，因其高性能、内存安全和并发支持。
- **哈希算法**：SHA-256，通过 sha2 库实现。
- **时间戳**：使用 chrono 库获取当前时间。
- **数据序列化**：使用 serde 和 serde_json 进行区块数据的序列化。

2. 系统架构

1. **区块链结构**：
    - 区块链由多个区块组成，每个区块包含索引、时间戳、哈希值、前一区块哈希和数据。
    - 使用 Vec<Block> 存储区块。
2. **创世区块生成**：
    - 创建一个索引为 0 的区块，前一区块哈希为空。
    - 计算区块的哈希值并存储。
3. **挖矿算法**：
    - 实现简单的挖矿算法，通过不断尝试不同的随机数（nonce）来生成满足难度要求的哈希值。

3. 实现步骤

1. **环境搭建**：
    - 安装 Rust 编译器。
    - 创建 Rust 项目并添加依赖。
2. **代码实现**：
    - 定义 Block 和 Blockchain 数据结构。
    - 实现创世区块的生成和哈希计算。
    - 提供命令行接口，展示区块信息。
3. **测试与验证**：
    - 单节点测试，验证创世区块的生成和展示。
    - 确保哈希值符合设计要求。

4. 代码示例

以下是生成创世区块的核心代码示例：

```rust
use bincode::serialize;
use sha2::{Sha256, Digest};
pub type Result<T> = std::result::Result<T, Box<dyn std::error::Error>>;
use hex;
use chrono::{DateTime, Utc, TimeZone};
use serde::{Serialize, Deserialize};

#[derive(serde::Serialize, Debug, Clone)]
pub struct Block {
    index: u32,
    #[serde(with = "chrono::serde::ts_milliseconds")]
    timestamp: DateTime<Utc>,
    data: String,
    prev_hash: String,
    hash: String,
    nonce: u32,
}

impl Block {
    pub fn new_block(index: u32, data: String, prev_hash: String, nonce: u32) -> Result<Block> {
        let hash = String::new();
        let timestamp: DateTime<Utc> = Utc::now();
        let mut block = Block {
            index,
            timestamp,
            data,
            prev_hash,
            hash,
            nonce,
        };

        block.validate_block()?;
        Ok(block)
```

```rust
        block.validate_block()?;
        Ok(block)
    }
    pub fn new_genesis_block(data: String) -> Block {
        Block::new_block(0, data, String::new(),0).unwrap()
    }
    pub fn calculate_hash(&self) -> String {
        let mut hasher = Sha256::new();
        let input = &self.hash_data().unwrap()[..];
        hasher.update(input);
        let result = hasher.finalize();
        hex::encode(result)
    }
}

#[derive(serde::Serialize, Debug, Clone)]
pub struct Blockchain {
    pub blocks: Vec<Block>,
}

impl Blockchain {
    pub fn new() -> Blockchain {
        let mut blockchain = Blockchain { blocks: Vec::new() };
        blockchain.blocks.push(Block::new_genesis_block(String::from("Genesis Block")));
        blockchain
    }
}

fn main() {
    let mut blockchain = Blockchain::new();
    blockchain.add_block(Block::new_block(1, String::from("Block 1"), blockchain.blocks.last().unwrap().get_hash(),0).unwrap())
    blockchain.display();
}
```