程序员老廖的个人空间-程序员老廖个人主页-哔哩哔哩视频 (bilibili.com)

1 为什么是全网最全C++云存储项目

- 支持注册登录
- 支持文件管理
- 支持视频预览
- 支持图片预览
- 支持文件分享
- 优化了大文件上传
- 支持断点下载

2 从这个项目你可以学到什么

这个项目包含的技术栈

2.1 C++ 11及以上语法的经典应用

当前项目使用到的主要C++11特性。

2.1.1 智能指针

- `std::shared_ptr`：共享所有权的智能指针
- `std::weak_ptr`：弱引用，配合shared_ptr使用
- `std::unique_ptr`：独占所有权的智能指针

```
std::shared_ptr<Connection> conn(new Connection());
std::weak_ptr<Connection> weakConn = conn;
```

2.1.2 右值引用和移动语义

- 使用`&&`表示右值引用
- `std::move()`将左值转换为右值
- 移动构造函数和移动赋值运算符

```
class Buffer {
    Buffer(Buffer&& other) noexcept;  // 移动构造函数
    Buffer& operator=(Buffer&& other) noexcept;  // 移动赋值运算符
};
```

2.1.3 Lambda表达式

- 用于创建匿名函数对象
- 常用于回调函数

```
auto callback = [](const TcpConnectionPtr& conn) {
    // 处理连接
};
```

2.1.4 auto关键字

- 自动类型推导
- 简化代码，提高可读性

```
auto result = calculateResult();
auto it = container.begin();
```

2.1.4 nullptr

- 替代NULL的空指针常量
- 类型安全

```
void* ptr = nullptr;
```

2.1.6 范围for循环

- 简化容器遍历

```
for (const auto& item : container) {
    // 处理item
}
```

2.1.7 线程支持

- `std::thread`：线程类
- `std::mutex`：互斥量
- `std::condition_variable`：条件变量
- `std::lock_guard`和`std::unique_lock`：RAII锁管理

```
std::mutex mtx;
std::lock_guard<std::mutex> lock(mtx);
```

2.1.8 原子操作

- `std::atomic`：原子类型
- 无锁编程支持

```
std::atomic<int> counter{0};
counter++;
```

2.1.9 时间工具

- `std::chrono`：时间库
- 高精度时间点和时间段

```
using namespace std::chrono;
auto now = system_clock::now();
```

2.1.10 函数对象包装器

- `std::function`：通用函数包装器
- `std::bind`：函数适配器

```
std::function<void(const TcpConnectionPtr&)> callback;
```

这些C++11特性在MyMuduo中的应用：

1. 智能指针用于管理TCP连接和定时器等资源
2. 移动语义用于优化Buffer的数据传输
3. Lambda表达式用于注册回调函数
4. 原子操作用于计数器和标志位
5. 线程支持用于实现EventLoop和ThreadPool

2.2 线程+线程池封装

- Thread 基于pthread封装线类
- ThreadPool 基于Thread 封装线程池

2.3 高性缓存/日志/定时器的设计

- Buffer类 高性能缓存
- AsyncLogging 高性能异步日志
- Timer 高性能定时器

2.4 高性能C++网络框架设计

支持多种网络模型：

1. 单 epoll 模型
2. 单 epoll + 线程池模型
3. main reactor + sub reactor 模型
4. main reactor + sub reactor + 线程池模型

2.4.1 单 Reactor 模型 (test_reactor_single.cc)

- 单个 Reactor（EventLoop）线程负责所有的事件监听和处理
- 所有的 IO 操作和业务处理都在同一个线程中完成

```
                Main Thread (Reactor)
                        |
                  EventLoop/Epoll
                        |
          ---------------------------------
         |              |                  |
Accept新连接      读写网络数据          处理业务逻辑
```

2.4.2 单 Reactor + 线程池模型

- 单个 Reactor 线程处理所有的 IO 事件
- 线程池处理耗时的业务逻辑
- IO 和业务处理解耦

```
Main Thread (Reactor)              Thread Pool
       |                               |
  EventLoop/Epoll              ----------------
       |                       |   |    |    |
   IO Events                Worker Worker Worker Worker
       |                       |   |    |    |
Accept/Read/Write         Business Logic Processing
```

2.4.3 多Reactors 模型

- 主 Reactor 只负责接收新连接
- 多个 Sub Reactor 负责处理 IO 事件
- 通过多线程提高 IO 处理性能

```
                Main Reactor Thread
                       |
                   Accept新连接
                       |
        ------------------------------
        |              |              |
   Sub Reactor 1  Sub Reactor 2  Sub Reactor 3
        |              |              |
    IO Events      IO Events      IO Events
```

2.4.4 多Reactor + 线程池模型

- 主 Reactor 负责接收新连接
- 多个 Sub Reactor 处理 IO 事件
- 线程池处理业务逻辑
- 最大程度的并发处理

```
            Main Reactor Thread
                   |
               Accept新连接
                   |
    ---------------------------------
    |              |                |
Sub Reactor 1  Sub Reactor 2    Sub Reactor 3
    |              |                |
IO Events      IO Events        IO Events
    |              |                |
    ----------Thread Pool------------
    |      |     |     |     |
Worker  Worker  Worker  Worker  Worker
```

2.5 高性能HTTP组件设计

优化内容：

- 在muduo http框架上优化大文件上传。
- 优化文件断点下载（支持range 下载指定范围的数据）
- session权限校验

涉及的类文件：

- HttpContext
- HttpRequest
- HttpResponse

- HttpServer
- FileUploadContext 处理文件上传上下文
- FileDownloadContext 处理文件下载上下文

2.5 数据库操作

数据库（用户注册/登录，文件管理，分享文件管理等表的设计）

- users用户表
- sessions 保存用户会话用于HTTP API校验
- files 保存个人文件
- file_shares 保存文件分享记录

```
-- 删除数据库
DROP DATABASE file_manager;
-- 创建数据库
CREATE DATABASE IF NOT EXISTS file_manager DEFAULT CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;

USE file_manager;

-- 创建用户表
CREATE TABLE IF NOT EXISTS users (
    id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) NOT NULL UNIQUE,
    password VARCHAR(64) NOT NULL,
    email VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_username (username)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- 创建会话表
CREATE TABLE IF NOT EXISTS sessions (
    id INT PRIMARY KEY AUTO_INCREMENT,
    session_id VARCHAR(32) NOT NULL UNIQUE,
    user_id INT NOT NULL,
    username VARCHAR(50) NOT NULL,
    expire_time TIMESTAMP NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    INDEX idx_session_id (session_id),
    INDEX idx_user_id (user_id),
    INDEX idx_expire_time (expire_time),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```sql
-- 创建文件表
CREATE TABLE IF NOT EXISTS files (
    id INT PRIMARY KEY AUTO_INCREMENT,
    filename VARCHAR(255) NOT NULL,
    original_filename VARCHAR(255) NOT NULL,
    file_size BIGINT UNSIGNED NOT NULL,
    file_type VARCHAR(50),
    user_id INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    INDEX idx_filename (filename),
    INDEX idx_user_id (user_id),
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- 创建文件分享表
CREATE TABLE IF NOT EXISTS file_shares (
    id INT PRIMARY KEY AUTO_INCREMENT,
    file_id INT NOT NULL,
    owner_id INT NOT NULL,
    shared_with_id INT,
    share_type ENUM('private', 'public', 'protected', 'user') NOT NULL,
    share_code VARCHAR(32) NOT NULL,
    expire_time TIMESTAMP NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    extract_code VARCHAR(6),
    INDEX idx_file_id (file_id),
    INDEX idx_owner_id (owner_id),
    INDEX idx_shared_with_id (shared_with_id),
    INDEX idx_share_code (share_code),
    INDEX idx_expire_time (expire_time),
    FOREIGN KEY (file_id) REFERENCES files(id) ON DELETE CASCADE,
    FOREIGN KEY (owner_id) REFERENCES users(id) ON DELETE CASCADE,
    FOREIGN KEY (shared_with_id) REFERENCES users(id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

3 把项目跑起来

3.1 导入数据库

```
// 进入项目目录
cd lesson29_file_manager
// 使用用户名 密码导入数据库
mysql -u username -ppassword < file_manager.sql
```

比如我username是root，password是123456，即是

```
mysql -u root -p123456 < file_manager.sql
```

然后登录数据库查看是否导入正常。

选择数据库

```
mysql> use file_manager;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

显示当前数据库的表单

```
mysql> show tables;
+------------------------+
| Tables_in_file_manager |
+------------------------+
| file_shares            |
| files                  |
| sessions               |
| users                  |
+------------------------+
4 rows in set (0.00 sec)
```

3.2 编译项目和运行项目

```
cd lesson29_file_manager
mkdir build
cd build
cmake ..
make
运行：
./bin/http_upload
```

4 学习这个项目需要哪些技术栈基础

- C++11及以上常用语言特性
- C++ STL
- Linux多线程编程
- Linux网络编程
- Linux C++网络框架实现
- Linux Http协议
- MySQL数据库知识

5 项目特色

- 大文件上传处理
- 支持文件断点下载

6 附录

将MP4文件的`moov`原子（metadata box）移动到文件头部（这有助于视频流式传输和快速播放），可以使用FFmpeg的`-movflags faststart`选项。

ffmpeg -i input.mp4 -movflags faststart -c copy output.mp4

```
-movflags faststart
```

ffmpeg -i C++Linux项目推荐-进阶版Webserver-Web聊天室+MySQL+Redis.mp4 -movflags faststart -c copy C++Linux项目推荐-进阶版Webserver-Web聊天室+MySQL+Redis2.mp4