redis 安装与使用

版权说明

对比特项目感兴趣，可以联系这个微信。

代码 & 板书链接

https://gitee.com/bitedu-tech/cpp-chatsystem

介绍

Redis（Remote Dictionary Server）是一个开源的高性能键值对（key-value）数据库。它通常用作数据结构服务器，因为除了基本的键值存储功能外，Redis 还支持多种类型的数据结构，如字符串（strings）、哈希（hashes）、列表（lists）、集合（sets）、有序集合（sorted sets）以及范围查询、位图、超日志和地理空间索引等。

以下是 Redis 的一些主要特性：

1. **内存中数据库**：Redis 将所有数据存储在内存中，这使得读写速度非常快。
2. **持久化**：尽管 Redis 是内存数据库，但它提供了持久化选项，可以将内存中的数据保存到磁盘上，以防系统故障导致数据丢失。
3. **支持多种数据结构**：Redis 不仅支持基本的键值对，还支持列表、集合、有序集合等复杂的数据结构。
4. **原子操作**：Redis 支持原子操作，这意味着多个操作可以作为一个单独的原子步骤执行，这对于并发控制非常重要。
5. **发布/订阅功能**：Redis 支持发布订阅模式，允许多个客户端订阅消息，当消息发布时，所有订阅者都会收到消息。
6. **高可用性**：通过 Redis 哨兵（Sentinel）和 Redis 集群，Redis 可以提供高可用性和自动故障转移。
7. **复制**：Redis 支持主从复制，可以提高数据的可用性和读写性能。
8. **事务**：Redis 提供了事务功能，可以保证一系列操作的原子性执行。
9. **Lua 脚本**：Redis 支持使用 Lua 脚本进行复杂的数据处理，可以在服务器端执行复杂的逻辑。
10. **客户端库**：Redis 拥有丰富的客户端库，支持多种编程语言，如 Python、Ruby、Java、C# 等。
11. **性能监控**：Redis 提供了多种监控工具和命令，可以帮助开发者监控和优化性能。
12. **易于使用**：Redis 有一个简单的配置文件和命令行界面，使得设置和使用变得容易。

Redis 广泛用于缓存、会话存储、消息队列、排行榜、实时分析等领域。由于其高性能和灵活性，Redis 成为了现代应用程序中非常流行的数据存储解决方案之一。

安装

使用 apt 安装

```Shell
apt install redis -y
```

支持远程连接

修改 `/etc/redis/redis.conf`

- 修改 `bind 127.0.0.1` 为 `bind 0.0.0.0`
- 修改 `protected-mode yes` 为 `protected-mode no`

```Plain Text
# By default, if no "bind" configuration directive is specified,
Redis listens
# for connections from all the network interfaces available on the
server.
# It is possible to listen to just one or multiple selected
interfaces using
# the "bind" configuration directive, followed by one or more IP
addresses.
#
# Examples:
#
# bind 192.168.1.100 10.0.0.1
# bind 127.0.0.1 ::1
#
# ~~~ WARNING ~~~ If the computer running Redis is directly
exposed to the
# internet, binding to all the interfaces is dangerous and will
expose the
# instance to everybody on the internet. So by default we
uncomment the
# following bind directive, that will force Redis to listen only
into
# the IPv4 loopback interface address (this means Redis will be
able to
# accept connections only from clients running into the same
computer it
# is running).
#
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE
INTERFACES
```

```
# JUST COMMENT THE FOLLOWING LINE.
#
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~
# bind 127.0.0.1   # 注释掉这行
bind 0.0.0.0       # 添加这行

protected-mode no  # 把 yes 改成 no
```

控制 Redis 启动

启动 Redis 服务

```
Plain Text
service redis-server start
```

停止 Redis 服务

```
Plain Text
service redis-server stop
```

重启 Redis 服务

```
Plain Text
service redis-server restart
```

装 redis-plus-plus

C++ 操作 redis 的库有很多. 咱们此处使用 redis-plus-plus.

这个库的功能强大, 使用简单.

Github 地址: https://github.com/sewenew/redis-plus-plus

安装 hiredis

redis-plus-plus 是基于 hiredis 实现的.

> hiredis 是一个 C 语言实现的 redis 客户端.

因此需要先安装 hiredis. 直接使用包管理器安装即可.

```
Plain Text
apt install libhiredis-dev
```

下载 redis-plus-plus 源码

```
Bash
git clone https://github.com/sewenew/redis-plus-plus.git
```

编译/安装 redis-plus-plus

使用 cmake 构建

```
Bash
cd redis-plus-plus

mkdir build

cd build

cmake ..

make

make install    # 这一步操作需要管理员权限. 如果是非 root 用户, 使用 sudo make install 执行.
```

构建成功后, 会在 `/usr/local/include/` 中多出 `sw` 目录, 并且内部包含 redis-plus-plus 的一系列头文件.

会在 `/usr/local/lib/` 中多出一系列 `libredis` 库文件.

接口

redis 本身支持很多数据类型的键值对，但是在聊天室项目中只涉及到了字符串键值对的操作，因此这里主要介绍字符串键值对的基础操作。

```
C++
namespace sw {
namespace redis {
    struct ConnectionOptions {
        std::string host;
        int port = 6379;
```

```
        std::string path;
        std::string user = "default";
        std::string password;
        int db = 0; // 默认 0 号库
        bool keep_alive = false;
    }
    struct ConnectionPoolOptions {
        std::size_t size = 1; //最大连接数量
    }
    class Redis {
        // uri e.g 'tcp://127.0.0.1:6379'
        explicit Redis(const std::string &uri)
        explicit Redis(const ConnectionOptions &connection_opts,
            const ConnectionPoolOptions &pool_opts = {})
        //删除当前库中所有数据
        void flushdb(bool async = false);
        //删除指定键值对
        long long del(const StringView &key);
        //判断指定键值对是否存在
        long long exists(const StringView &key);
        //获取一个 string 键值对
        OptionalString get(const StringView &key);
        //存放一个 string 键值对，且设置过期时间-毫秒
        bool set(const StringView &key,
                const StringView &val,
                const std::chrono::milliseconds &ttl =
                    std::chrono::milliseconds(0), // 0 表示不设置超
时
                UpdateType type = UpdateType::ALWAYS);
        void setex(const StringView &key,
                long long ttl,
                const StringView &val);
        //向一个列表中尾插/头插 string 键值对
        long long rpush(const StringView &key, const StringView
&val);
        long long lpush(const StringView &key, const StringView
&val);
        long long rpush(const StringView &key,
                Input first, Input last);
        // std::vector<std::string> elements;
        // redis.lrange("list", 0, -1,
std::back_inserter(elements));
        void lrange(const StringView &key,
```

```
            long long start, long long stop, Output output);
    }
}
}
```

使用

这里只进行字符串键值对的增删改查操作以及数据的生命周期设置。

```C++
#include <sw/redis++/redis.h>
#include <iostream>
#include <string>
#include <thread>
#include <gflags/gflags.h>

DEFINE_bool(redis_keep_alive, true, "是否保持长连接");
DEFINE_int32(redis_db, 0, "redis 库号");
DEFINE_int32(redis_port, 6379, "redis 服务器端口");
DEFINE_string(redis_host, "127.0.0.1", "redis 服务器 IP 地址");

std::shared_ptr<sw::redis::Redis> predis;
void add() {
    predis->set("用户会话 1", "用户 ID1");
    predis->set("用户会话 2", "用户 ID2",
        std::chrono::milliseconds(1000)); //设置 1000ms 过期时间
    predis->set("用户会话 3", "用户 ID3");
    predis->set("用户会话 4", "用户 ID4");
    predis->set("用户会话 5", "用户 ID5");
}
void get() {
    auto res1 =  predis->get("用户会话 1");
    if (res1) std::cout << *res1 << std::endl;

    auto res2 =  predis->get("用户会话 2");
    if (res2) std::cout << *res2 << std::endl;

    auto res3 =  predis->get("用户会话 3");
    if (res3) std::cout << *res3 << std::endl;

    auto res4 =  predis->get("用户会话 4");
    if (res4) std::cout << *res4 << std::endl;
```

```
    auto res5 =  predis->get("用户会话 5");
    if (res5) std::cout << *res5 << std::endl;
}
void update() {
    predis->set("用户会话 1", "用户 ID 变成 31");
    predis->set("用户会话 4", "用户 ID 变成 41",
        std::chrono::milliseconds(1000));
    predis->del("用户会话 5");
}

void push_test() {
    predis->rpush("群聊会话 1", "成员 1");
    predis->rpush("群聊会话 1", "成员 2");
    predis->rpush("群聊会话 1", "成员 3");
    predis->rpush("群聊会话 1", "成员 4");
    predis->rpush("群聊会话 1", "成员 5");

    predis->rpush("群聊会话 2", "成员 6");
    predis->rpush("群聊会话 2", "成员 7");
    predis->rpush("群聊会话 2", "成员 8");
    predis->rpush("群聊会话 2", "成员 9");
    predis->rpush("群聊会话 2", "成员 0");

    std::vector<std::string> res;
    predis->lrange("群聊会话 1", 0, -1, std::back_inserter(res));
    for (const auto &r : res) {
        std::cout << r << std::endl;
    }
}
int main()
{
    sw::redis::ConnectionOptions opts;
    opts.host = FLAGS_redis_host;
    opts.port = FLAGS_redis_port;
    opts.db = FLAGS_redis_db;
    opts.keep_alive  = FLAGS_redis_keep_alive;
    predis = std::make_shared<sw::redis::Redis>(opts);
    std::cout << "--------add-------\n";
    add();
    std::cout << "--------get-------\n";
    get();
    std::cout << "--------2s get-------\n";
```

```
    std::this_thread::sleep_for(std::chrono::seconds(2));
    get();
    std::cout << "--------update-------\n";
    update();
    std::cout << "--------2s get-------\n";
    std::this_thread::sleep_for(std::chrono::seconds(2));
    get();

    std::cout << "--------push-------\n";
    push_test();
    return 0;
}
```

```
C++
main : main.cc
    g++ -std=c++17 $^ -o $@ -lredis++ -lhiredis -lgflags -pthread
```