C++ 使用 redis

1. 安装 redis

```
# 安装 redis-server, redis c 客户端函数库
pacman -S redis hiredis
```

2. 启用 redis

```
# 启用 redis-server 服务器
systemctl start redis.service
# 查看 redis-server status
systemctl status redis.service
```

3. 简单体验 redis

```
# redis 自带一个简单的 redis-client, 可以简单的体验一下
# port 默认是 6379,
redis-cli -h hostname -p port
```

简单的基本命令 :

```
# 测试连接
127.0.0.1:6379> PING
PONG
# 设置 key: "microsoft"  value: "windows"
127.0.0.1:6379> SET microsoft windows_xp
OK
# 获取 key: "microsoft"
127.0.0.1:6379> GET microsoft
"windows_xp"
# 获取不存在的值
127.0.0.1:6379> GET apple
(nil)
# 退出 redis-client
127.0.0.1:6379> QUIT
```

4. redis 介绍 :

redis 是一款开源的高性能内存 KV 数据库, 你可以简单的将他理解为一个 c++ std::unorderedmap<key, value>, 他的 key value, 支持多种数据类型. 且支持很多强大的功能.

5. redis 配置

1. redis 设置用户登陆密码

修改 /etc/redis/redis.conf, 找到 #requirepass foobared 这一行, 去掉注释, 将 foobared 改为你要的密码. 之后 **重启 redis 服务**.

之后再次使用 redis-client 连接, 虽然可以连接, 但是没有操作权限.

```
127.0.0.1:6379:PING
(error) NOAUTH Authentication required
```

- 在 redis-client 加入 你的密码

```
redis-client -a yourpassword
```

- 在命令行加入 密码

```
127.0.0.1:6379> AUTH yourpassword
```

认证成功即可正常使用.

2. 由于 redis 是内存数据库, 普通的内存在掉电时数据全部丢失, 所以 redis 提供了两种数据持久化的方式, 将数据刷到磁盘中, AOF / RDB. 这里只介绍 AOF.

   AOF 是 appendonlyfile 的缩写, 将你对数据库进行写入的命令记录到 AOF 文件中, 宕机之后，可以从 AOF 文件中读取命令来恢复数据.

   配置 AOF

   - 在 redis.conf 中, 将 appendonly 打开, 表示开启 redis AOF, 这里有三种持久化方式:
     - appendfsync always (不会丢失数据, 但性能差)
     - appendfsync everysec (每秒记录, 可能会丢失 1s 内的数据) (推荐)
     - appendfsync no (将缓存回写机制交给系统, linux 默认 30s 写硬盘)

     使用哪个请将其他另外两个注释即可.

6. c++ 使用 redis

   1. 开头中所提到的 hiredis就是 使用 redis c 风格的函数库

      ```
      #include <hiredis/hiredis.h>
      ```

   2. 连接 redis

      ```
      redisContext* redisConnect(const char* ip, int port);
      // 带有超时的连接
      redisContext* redisConnectWithTimeout(const char* ip, int port, const struct timeout);
      ```

      redisConnect 返回的一个 redis-server 句柄

      后续所有的操作都通过该句柄来完成.

      使用完成之后, 请释放句柄.

      ```
      void redisFree(redisContext* con);
      ```

   3. 操作 redis

      ```
      void* redisCommand(redisContext* c, const char* format, ...);
      ```

      通过该命令即可操作redis, 该函数错误返回 NULL.

      正常返回一个 redisReply, 所以返回值需要转换.

      每次调用 redisCommand, 记得 释放资源,

```
void freeReplyObject(void* reply);
```

4. 你可能会用到的命令 :

   redis hash 是一个 string 类型的 字段 和 value 的映射表, hash 特别适合存储对象.

```
127.0.0.1:6379> HMSET xiao_ming name "xiao_ming" age "12" number "110"
OK
127.0.0.1:6379> HGETALL xiao_ming
1) "name"
2) "xiao_ming"
3) "age"
4) "12"
5) "number"
6) "110"
# 删除 key
127.0.0.1:6379> DEL key_name
# 更改 key_name
127.0.0.1:6379> RENAME key_name
# 查询 key_name 是否存在
127.0.0.1:6379> EXISTS key_name
# 如果 key 是数字类型的, 那么可以使用 incr 命令每次递增
127.0.0.1:6379> INCR key_name
# 批量 set
127.0.0.1:6379> MSET k_1 v_1 k_2 v_2 ...
# 追加 key 的内容
127.0.0.1:6379> APPEND key_name append_string
# 列表 (list) 集合 (set (无序)) (有序集合 (sorted set))
```

   关于其他的命令, 你可以去 https://redis.io/commands/ 学习.

7. 关于项目

   由于 hiredis 库是标准的 c 函数接口, 禁止直接使用原生的接口, 需要抽象一系列的类. 由类来向外提供方法.

   当然这里在 序列化 和 反序列化 还是很麻烦.

   我的建议是是使用 JSON 库 和 Redis 结合起来完成会比较方便.

   推荐 2 个 C++ JSON 库

   - https://github.com/nlohmann/json (好用, 用了大量运算符重载, 使用起来很舒服. 速度一般)
   - https://github.com/Tencent/rapidjson/ (Tencent 开源的一款 JSON 库, 速度较快, 但听说使用体验糟糕)

   当然 Redis 也有 JSON 模块支持.

   - https://github.com/RedisJSON/RedisJSON
   - 使用请看 : https://www.jianshu.com/p/5ca12b61e85b 和 https://redis.io/docs/stack/json/

   关于怎么将 JSON 库 怎么和 redis 结合起来使用 , 我相信诸君一定很容易想到吧.

8. 一个简单的 redis 使用demo

```
#include <iostream>
#include <hiredis/hiredis.h>
#include <json/json.h>
#include <json/assertions.h>
#include <stdio.h>

int main(void) {
  const char* RedisIP = "127.0.0.1";
  //* 默认端口是 6379
```

```
  int RedisPort = 6379;
  struct timeval timeout = {1, 500000};
  // redisConnectWithTimeout 连接 redis-server
  // 并返回一个句柄
  redisContext* conn = redisConnectWithTimeout(RedisIP, RedisPort, timeout);
  if (conn == nullptr || conn->err) {
    printf("errno!, can not connect redis server ip: %s, port: %d, err: %d", RedisIP, RedisPort, conn->err);
  }
  const char* password = "123";
  redisReply* reply = static_cast<redisReply*>(redisCommand(conn, "AUTH %s", password));
  if (reply == nullptr) {
    printf("errno! redisCommand: AUTH!\n");
  } else {
    printf("AUTH: %s\n", reply->str);
  }
  // 每次使用完记得调用 freeReplyObject
  freeReplyObject(reply);
  reply = static_cast<redisReply*>(redisCommand(conn, "SET microsoft windows_xp"));
  if (reply == nullptr) {
    printf("errno! redisCommand: SET!\n");
  } else {
    printf("%s\n", reply->str);
  }
  freeReplyObject(reply);
  reply = static_cast<redisReply*>(redisCommand(conn,"GET microsoft"));
  if (reply == nullptr) {
    printf("errno! redisCommand: GET microsoft!\n");
  } else {
    printf("%s\n", reply->str);
  }
  freeReplyObject(reply);
  // 断开连接需要释放一些资源.
  redisFree(conn);
  return 0;
}
```

编译运行：

```
g++ redis_test.cpp -o redis_test -O2 -Wall -lhiredis
./redis_test
```

9. 一个nlohmann_json demo

```
#include <iostream>
#include <string>
// 使用 nlohmann_json 库只需包含即可，无需链接
#include <nlohmann/json.hpp>

namespace OVL {
// 注意 From_Json 函数 To_Json 对象必须在同一 namespace 中
class User {
public:
  std::string name;
  uint64_t age;
  std::string number;
  std::string password;
```

```
  //  JSON => class
  static void From_Json (const nlohmann::json& jn, User& user) {
    jn.at("name").get_to(user.name);
    jn.at("age").get_to(user.age);
    jn.at("number").get_to(user.number);
    jn.at("password").get_to(user.password);
  }
  // class => JSON
  static void To_Json (nlohmann::json& jn, const User& user) {
    jn = nlohmann::json{
      {"name", user.name},
      {"age", user.age},
      {"number", user.number},
      {"password", user.password}
    };
  }
  void PrintInfo() {
    printf("name = %s\nage = %ld \nnumber = %s \npassword = %s\n", \
        name.c_str(), age, number.c_str(), password.c_str());
  }
};
}

int main(void) {
  // 1.  在序列化字符串之后加入 _json后缀.
  nlohmann::json jn_1 = "{\"name\": \"xiao_hong\",  \
                          \"age\": 12,  \
                          \"number\": \"123\",  \
                          \"password\": \"456\" }"_json;
  OVL::User xiao_hong;
  OVL::User::From_Json(jn_1, xiao_hong);
  xiao_hong.PrintInfo();
  // 3. 直接构造
  nlohmann::json jn_3 {
    { "name", "xiao_lan"},
    { "age", 12},
    { "number", "123"},
    { "password", "123"}
  };
  // 2. 显示的调用 json::parse 推荐
  auto jn_2 = nlohmann::json::parse(R"(
                                      {
                                        "name": "xiao_ming",
                                        "age": 12,
                                        "number": "123",
                                        "password": "234"
                                      }
                                    )");
  OVL::User xiao_ming;
  OVL::User::From_Json(jn_2, xiao_ming);
  xiao_ming.PrintInfo();

  nlohmann::json xiao_ming_json;
  OVL::User::To_Json(xiao_ming_json, xiao_ming);
  // nlohmann::json.dump  函数返回一个 json 字符串
  std::string xiao_ming_json_string = xiao_ming_json.dump();
  printf("%s\n", xiao_ming_json_string.c_str());
```

```
    /*
    * 当然我们也可以直接更改json 对象的某一个字段
    *   at 方法，当字段不存在时会抛出一个 out_of_range异常
    */
    xiao_ming_json["name"] = "xiao_gang";
    xiao_ming_json.at("name") = "xiao_gang";
    /*
     * 查找
     */
    auto find_obj = xiao_ming_json.find("password");
    // 直接访问
    std::cout << xiao_ming_json["password"] << " " << find_obj.key() << std::endl;

    return 0;
}
```

nlohmann_json 库的安装

```
pacman -S nlohmann-json
```