C++实现集群聊天服务器

> 本文档全部由施磊老师整理输出，不得任意拷贝，如要引用必须指明出处，违者必究
>
> 关注课堂地址：https://fixbug.ke.qq.com/

技术栈

- Json序列化和反序列化
- muduo网络库开发
- nginx源码编译安装和环境部署
- nginx的tcp负载均衡器配置
- redis缓存服务器编程实践
- 基于发布-订阅的服务器中间件redis消息队列编程实践
- MySQL数据库编程

- CMake构建编译环境
- Github托管项目

项目需求

1. 客户端新用户注册
2. 客户端用户登录
3. 添加好友和添加群组
4. 好友聊天
5. 群组聊天
6. 离线消息
7. nginx配置tcp负载均衡
8. 集群聊天系统支持客户端跨服务器通信

项目目标

1. 掌握服务器的网络I/O模块，业务模块，数据模块分层的设计思想
2. 掌握C++ muduo网络库的编程以及实现原理
3. 掌握Json的编程应用
4. 掌握nginx配置部署tcp负载均衡器的应用以及原理
5. 掌握服务器中间件的应用场景和基于发布-订阅的redis编程实践以及应用原理
6. 掌握CMake构建自动化编译环境
7. 掌握Github管理项目

开发环境

1. ubuntu linux环境
2. 安装Json开发库
3. 安装boost + muduo网络库开发环境，参考我的博客：https://blog.csdn.net/QIANGWEIYUAN/article/details/89023980
4. 安装redis环境
5. 安装mysql数据库环境
6. 安装nginx
7. 安装CMake环境

配置远程开发环境

> windows+vscode配置远程linux开发环境

参考博客：https://blog.csdn.net/qq756684177/article/details/94236990

1. linux系统运行sshd服务

2. 在vscode上安装Remote Development插件，其依赖插件会自动安装
3. 配置远程linux主机的信息
4. 在vscode上开发远程连接linux

> VS环境创建远程linux跨平台项目

参考博客：https://blog.csdn.net/QIANGWEIYUAN/article/details/89469717

> vscode在linux环境下直接开发

网络搜索关键词

下载

链接：https://pan.baidu.com/s/1kXyXy5ct1OmIxu8uSfjS1g 提取码：13gi

Json介绍

Json是一种轻量级的数据交换格式（也叫数据序列化方式）。Json采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 Json 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

一个优秀的Json三方库

JSON for Modern C++ 是一个由德国大牛 nlohmann 编写的在 C++ 下使用的 JSON 库。

具有以下特点

- 直观的语法
- 整个代码由一个头文件组成 json.hpp，没有子项目，没有依赖关系，没有复杂的构建系统，使用起来非常方便
- 使用 C++ 11 标准编写
- 使用 json 像使用 STL 容器一样
- STL 和 json 容器之间可以相互转换
- 严谨的测试：所有类都经过严格的单元测试，覆盖了 100% 的代码，包括所有特殊的行为。此外，还检查了 Valgrind 是否有内存泄漏。为了保持高质量，该项目遵循核心基础设施倡议(CII)的最佳实践

包含json头文件

在网络中，常用的数据传输序列化格式有XML，Json，ProtoBuf，在公司级别的项目中，大量的在使用ProtoBuf作为数据序列化的方式，以其数据压缩编码传输，占用带宽小，同样的数据信息，是Json的1/10，XML的1/20，但是使用起来比Json稍复杂一些，所以项目中我们选择常用的Json格式来打包传输数据。

下面列举一些项目中用到的有关Json数据的序列化和反序列化代码，仅供参考！JSON for Modern C++这个三方库的使用非常简单，如下所示：

```
#include "json.hpp"
using json = nlohmann::json;
```

Json数据序列化

就是把我们想要打包的数据，或者对象，直接处理成Json字符串。

1. 普通数据序列化

```
json js;
// 添加数组
js["id"] = {1,2,3,4,5};
// 添加key-value
js["name"] = "zhang san";
// 添加对象
js["msg"]["zhang san"] = "hello world";
js["msg"]["liu shuo"] = "hello china";
// 上面等同于下面这句一次性添加数组对象
js["msg"] = {{"zhang san", "hello world"}, {"liu shuo", "hello china"}};
cout << js << endl;
```

上面js对象的序列化结果是：

{"id":[1,2,3,4,5],msg":{"liu shuo":"hello china","zhang san":"hello world"},"name":"zhang san"}

2. 容器序列化

```
json js;
// 直接序列化一个vector容器
vector<int> vec;
vec.push_back(1);
vec.push_back(2);
vec.push_back(5);
js["list"] = vec;

// 直接序列化一个map容器
map<int, string> m;
m.insert({1, "黄山"});
m.insert({2, "华山"});
m.insert({3, "泰山"});
js["path"] = m;

cout<<js<<endl;
```

强大到直接把C++ STL中的容器内容可以直接序列化成Json字符串，上面代码打印如下：

{"list":[1,2,5],"path":[[1,"黄山"],[2,"华山"],[3,"泰山"]]}

Json数据反序列化

当从网络接收到字符串为Json格式，可以用JSON for Modern C++ 直接反序列化取得数据或者直接反序列化出对象，甚至容器，强大无比！

```
string jsonstr = js.dump();
cout<<"jsonstr:"<<jsonstr<<endl;

// 模拟从网络接收到json字符串，通过json::parse函数把json字符串专程json对象
json js2 = json::parse(jsonstr);

// 直接取key-value
string name = js2["name"];
cout << "name:" << name << endl;

// 直接反序列化vector容器
vector<int> v = js2["list"];
for(int val : v)
{
    cout << val << " ";
}
cout << endl;

// 直接反序列化map容器
map<int, string> m2 = js2["path"];
for(auto p : m2)
{
    cout << p.first << " " << p.second << endl;
}
cout << endl;
```

muduo网络库编程

muduo源码编译安装

muduo库源码编译安装和环境搭建，参考我的博客：
https://blog.csdn.net/QIANGWEIYUAN/article/details/89023980

基于muduo的客户端服务器编程

muduo网络库的编程很容易，要实现基于muduo网络库的服务器和客户端程序，只需要简单的组合TcpServer和TcpClient就可以，代码实现如下：

```
/*
```

```
服务器类，基于muduo库开发
*/
class ChatServer
{
public:
    // 初始化TcpServer
    ChatServer(muduo::net::EventLoop *loop,
        const muduo::net::InetAddress &listenAddr)
        :_server(loop, listenAddr, "ChatServer")
    {
        // 通过绑定器设置回调函数
        _server.setConnectionCallback(bind(&ChatServer::onConnection,
            this, _1));

        _server.setMessageCallback(bind(&ChatServer::onMessage,
            this, _1, _2, _3));

        // 设置EventLoop的线程个数
        _server.setThreadNum(10);
    }

    // 启动ChatServer服务
    void start()
    {
        _server.start();
    }

private:
    // TcpServer绑定的回调函数，当有新连接或连接中断时调用
    void onConnection(const muduo::net::TcpConnectionPtr &con);
    // TcpServer绑定的回调函数，当有新数据时调用
    void onMessage(const muduo::net::TcpConnectionPtr &con,
        muduo::net::Buffer *buf,
        muduo::Timestamp time);
private:
    muduo::net::TcpServer _server;
};
```

```
/*
客户端实现，基于C++ muduo网络库
*/
class ChatClient
{
public:
    ChatClient(muduo::net::EventLoop *loop,
        const muduo::net::InetAddress &addr)
        :_client(loop, addr, "ChatClient")
    {
        // 设置客户端TCP连接回调接口
        _client.setConnectionCallback(bind(&ChatClient::onConnection,
            this, _1));

        // 设置客户端接收数据回调接口
        _client.setMessageCallback(bind(&ChatClient::onMessage,
            this, _1, _2, _3));
    }
    // 连接服务器
```

```
    void connect()
    {
        _client.connect();
    }
private:
    // TcpClient绑定回调函数，当连接或者断开服务器时调用
    void onConnection(const muduo::net::TcpConnectionPtr &con);
    // TcpClient绑定回调函数，当有数据接收时调用
    void onMessage(const muduo::net::TcpConnectionPtr &con,
        muduo::net::Buffer *buf,
        muduo::Timestamp time);

    muduo::net::TcpClient _client;
};
```

用muduo中的线程池做计算任务

采用muduo进行服务器编程，如果遇到需要开辟多线程单独来处理复杂的计算任务或者其它阻塞任务等，不需要直接调用pthread_create来创建线程，muduo库提供的ThreadPool线程池管理类已经把Linux的线程创建完全封装起来了，如果想研究源码，可以剖析muduo中ThreadPool.cc和Thread.cc。

ThreadPool使用示例：

```
// 客户端输入界面，在单独的线程中接收用户输入进行发送操作
void userClient(const muduo::net::TcpConnectionPtr &con);
muduo::ThreadPool _pool;
```

```
/*
连接服务器成功后，开启和服务器的交互通信功能
*/
if (con->connected()) // 和服务器连接成功
{
    LOG_INFO << "connect server success!";
    // 启动线程专门处理用户的输入操作
    _pool.run(bind(&ChatClient::userClient, this, con));
}
else // 和服务器连接失败
{

}
```

muduo的日志系统

在开发软件产品过程中，日志的输出非常重要，可以记录很多软件运行过程中的信息，方便定位调试问题，跟踪统计信息等等，muduo库提供的日志级别有：

```
#define LOG_TRACE if (muduo::Logger::logLevel() <= muduo::Logger::TRACE) \
  muduo::Logger(__FILE__, __LINE__, muduo::Logger::TRACE, __func__).stream()
#define LOG_DEBUG if (muduo::Logger::logLevel() <= muduo::Logger::DEBUG) \
  muduo::Logger(__FILE__, __LINE__, muduo::Logger::DEBUG, __func__).stream()
#define LOG_INFO if (muduo::Logger::logLevel() <= muduo::Logger::INFO) \
```

```
  muduo::Logger(__FILE__, __LINE__).stream()
#define LOG_WARN muduo::Logger(__FILE__, __LINE__, muduo::Logger::WARN).stream()
#define LOG_ERROR muduo::Logger(__FILE__, __LINE__,
muduo::Logger::ERROR).stream()
#define LOG_FATAL muduo::Logger(__FILE__, __LINE__,
muduo::Logger::FATAL).stream()
#define LOG_SYSERR muduo::Logger(__FILE__, __LINE__, false).stream()
#define LOG_SYSFATAL muduo::Logger(__FILE__, __LINE__, true).stream()

// 示例：
LOG_INFO << "记录相应级别的日志信息";
```

muduo网络库的多线程模型

网络服务器编程常用模型

【方案1】： accept + read/write

不是并发服务器

【方案2】： accept + fork - process-pre-connection

适合并发连接数不大，计算任务工作量大于fork的开销

【方案3】：accept + thread thread-pre-connection

比方案2的开销小了一点，但是并发造成线程堆积过多

【方案4】： muduo的网络设计：reactors in threads - one loop per thread

方案的特点是one loop per thread，有一个main reactor负载accept连接，然后把连接分发到某个sub reactor（采用round-robin的方式来选择sub reactor），该连接的所用操作都在那个sub reactor所处的线程中完成。多个连接可能被分派到多个线程中，以充分利用CPU。

Reactor poll的大小是固定的，根据CPU的数目确定。

```
// 设置EventLoop的线程个数，底层通过EventLoopThreadPool线程池管理线程类EventLoopThread
_server.setThreadNum(10);
```

一个Base IO thread负责accept新的连接，接收到新的连接以后，使用轮询的方式在reactor pool中找到合适的sub reactor将这个连接挂载上去，这个连接上的所有任务都在这个sub reactor上完成。

如果有过多的耗费CPU I/O的计算任务，可以提交到创建的ThreadPool线程池中专门处理耗时的计算任务。

【方案5】： reactors in process - one loop pre process

nginx服务器的网络模块设计，基于进程设计，采用多个Reactors充当I/O进程和工作进程，通过一把accept锁，完美解决多个Reactors的“惊群现象”。

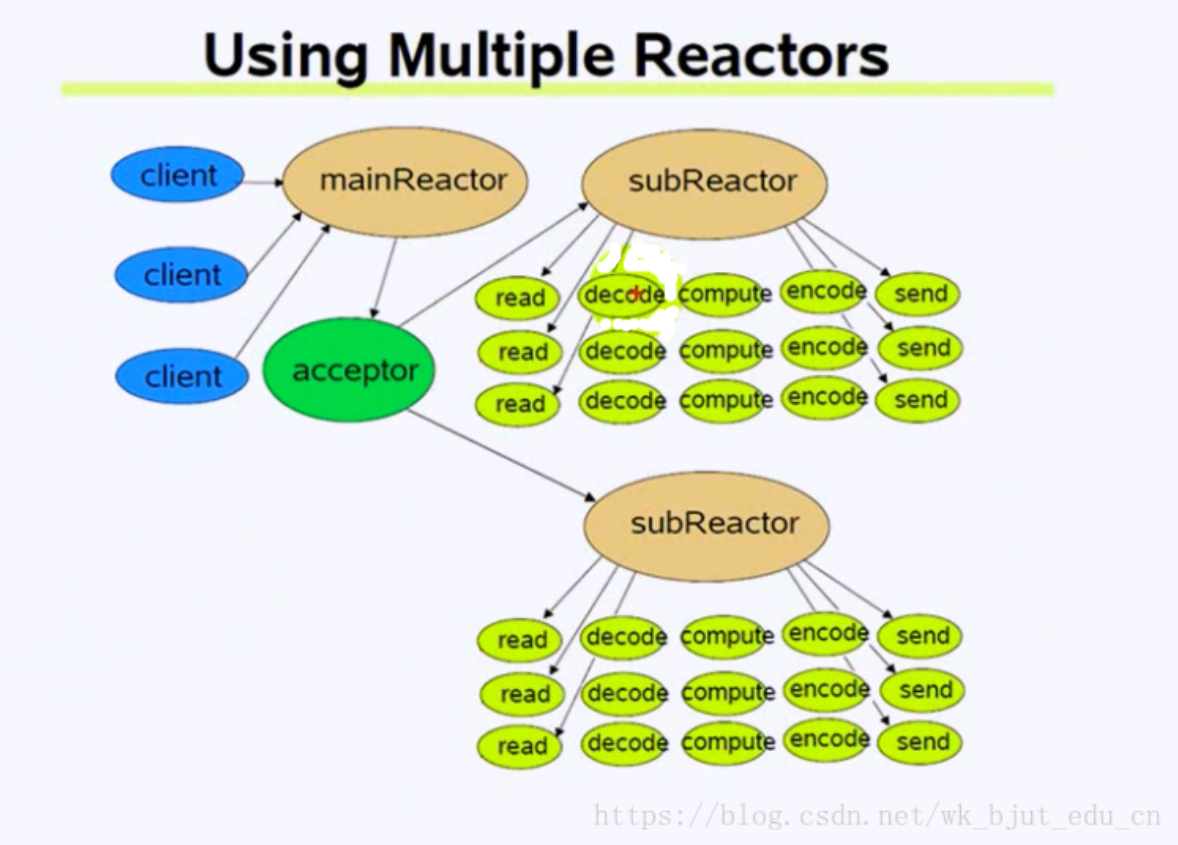
muduo中的reactor模型

reactor模型是什么？先看一下维基百科的标准解释：

> The reactor design pattern is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs. The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers.

从上面的描述，可以看出如下关键点：

1. 事件驱动（event handling）
2. 可以处理一个或多个输入源（one or more inputs）
3. 通过Service Handler同步的将输入事件（Event）采用多路复用分发给相应的Request Handler（多个）处理



建议大家打开muduo的源码，从TcpServer的start方法开始，阅读一下muduo库的源码实现，理解mainReactor和subReactor的工作原理，这样对于该项目的面试问题，也能更深入的去表达muduo相关的内容。

服务器集群

负载均衡器 - 一致性哈希算法

单台服务器受限于硬件资源，其性能是有上限的，当单台服务器不能满足应用场景的并发需求量时，就需要考虑部署多个服务器共同处理客户端的并发请求，但是客户端怎么知道去连接具体哪台服务器呢？此时就需要一台负载均衡器，通过预设的负载算法，指导客户端连接服务器。

负载均衡器有基于**客户端的负载均衡和服务器的负载均衡**。

普通的基于哈希的负载算法，并不能满足负载均衡所要求的**单调性**和**平衡性**，但**一致性哈希算法**非常好的保持了这两种特性，所以经常用在需要设计负载算法的应用场景当中。

具体的**负载均衡器的设计**和**一致性哈希算法**在课堂上进行详细讲解。

nginx配置tcp负载均衡

在服务器快速集群环境搭建中，都迫切需要一个能拿来即用的负载均衡器，nginx在1.9版本之前，只支持http协议web服务器的负载均衡，从1.9版本开始以后，nginx开始支持tcp的长连接负载均衡，但是nginx默认并没有编译tcp负载均衡模块，编写它时，需要加入--with-stream参数来激活这个模块。

> **nginx编译加入--with-stream参数激活tcp负载均衡模块**

nginx编译安装需要先安装pcre、openssl、zlib等库，也可以直接编译执行下面的configure命令，根据错误提示信息，安装相应缺少的库。

下面的make命令会向系统路径拷贝文件，需要在root用户下执行
tony@tony-virtual-machine:~/package/nginx-1.12.2# ./configure --with-stream
tony@tony-virtual-machine:~/package/nginx-1.12.2# make && make install

编译完成后，默认安装在了/usr/local/nginx目录。

tony@tony-virtual-machine:~/package/nginx-1.12.2$ cd /usr/local/nginx/
tony@tony-virtual-machine:/usr/local/nginx$ ls
conf html logs sbin

可执行文件在sbin目录里面，配置文件在conf目录里面。

```
root@tony-virtual-machine:/usr/local/nginx/sbin# ./nginx
root@tony-virtual-machine:/usr/local/nginx/sbin# netstat -tanp
激活Internet连接 (服务器和已建立连接的)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      8281/nginx: master
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      512/systemd-resolve
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1030/sshd
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN      2208/cupsd
tcp        0      0 127.0.0.1:6010          0.0.0.0:*               LISTEN      3149/sshd: tony@pts
tcp        0      0 127.0.0.1:33509         0.0.0.0:*               LISTEN      970/containerd
tcp        0      0 127.0.0.1:3306          0.0.0.0:*               LISTEN      1239/mysqld
tcp        0      0 127.0.0.1:6379          0.0.0.0:*               LISTEN      1033/redis-server 1
tcp        0     52 192.168.131.129:22      192.168.131.1:56664     ESTABLISHED 3022/sshd: tony [pr
tcp6       0      0 :::22                   :::*                    LISTEN      1030/sshd
tcp6       0      0 ::1:631                 :::*                    LISTEN      2208/cupsd
```

nginx -s reload 重新加载配置文件启动
nginx -s stop 停止nginx服务

> **nginx配置tcp负载均衡**

主要在conf目录里面配置nginx.conf文件，配置如下：

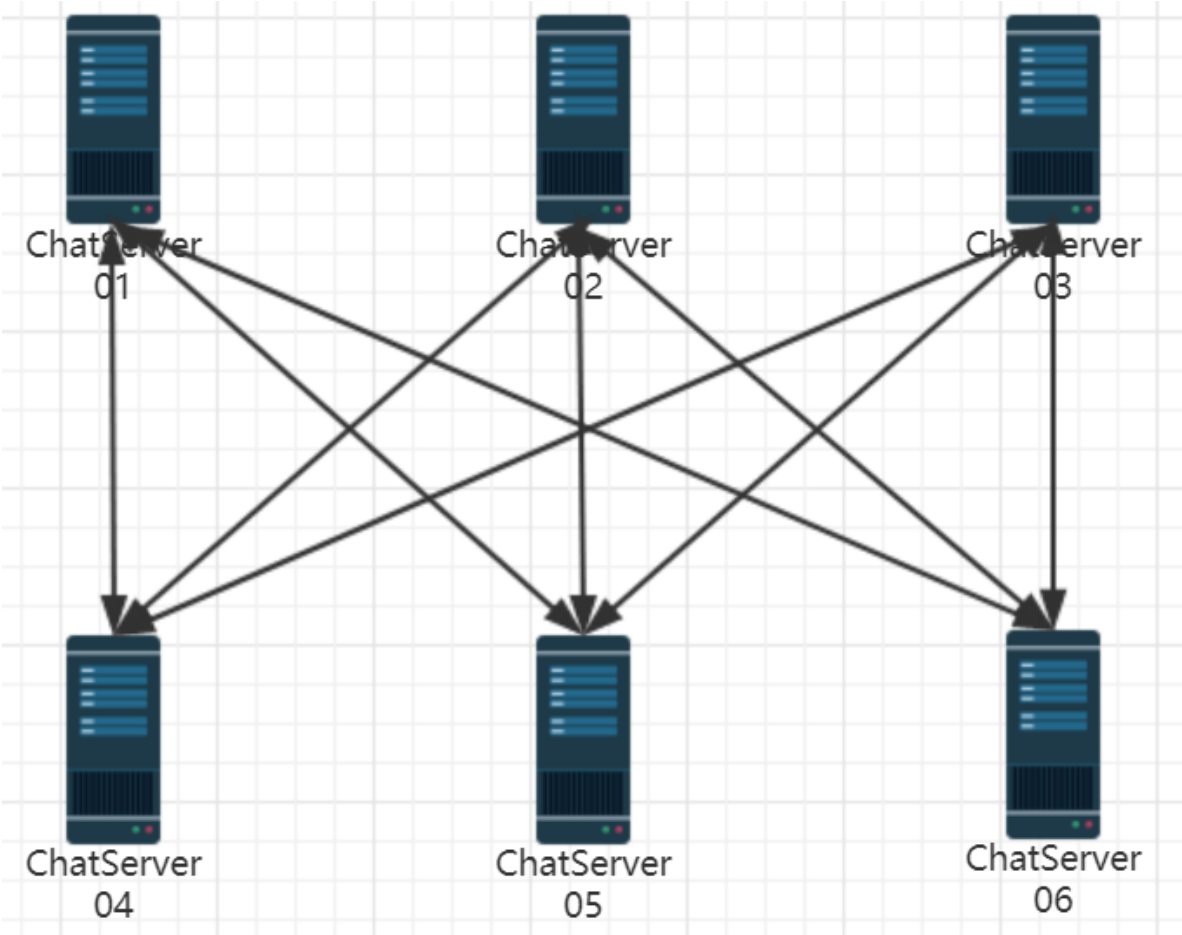
```
events {
    worker_connections  1024;
}

stream {
    upstream MyServer {
        hash $remote_addr consistent;
        server 192.168.131.129:6000 weight=1 max_fails=3 fail_timeout=30s;
        server 192.168.131.129:6002 weight=1 max_fails=3 fail_timeout=30s;
    }

    server {                                        tcp的负载均衡配置
        proxy_connect_timeout 1s;
        proxy_timeout 3s;
        listen 8000; # so_keepalive=on;
        proxy_pass MyServer;
        tcp_nodelay on;
    }                                               注意stream和http属于不同的配置模块，是同级别的，
}                                                   没有隶属关系。

http {
    include       mime.types;                       http的负载均衡
    default_type  application/octet-stream;

    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
```

配置完成后，./nginx -s reload平滑重启。

服务器中间件-基于发布-订阅的Redis

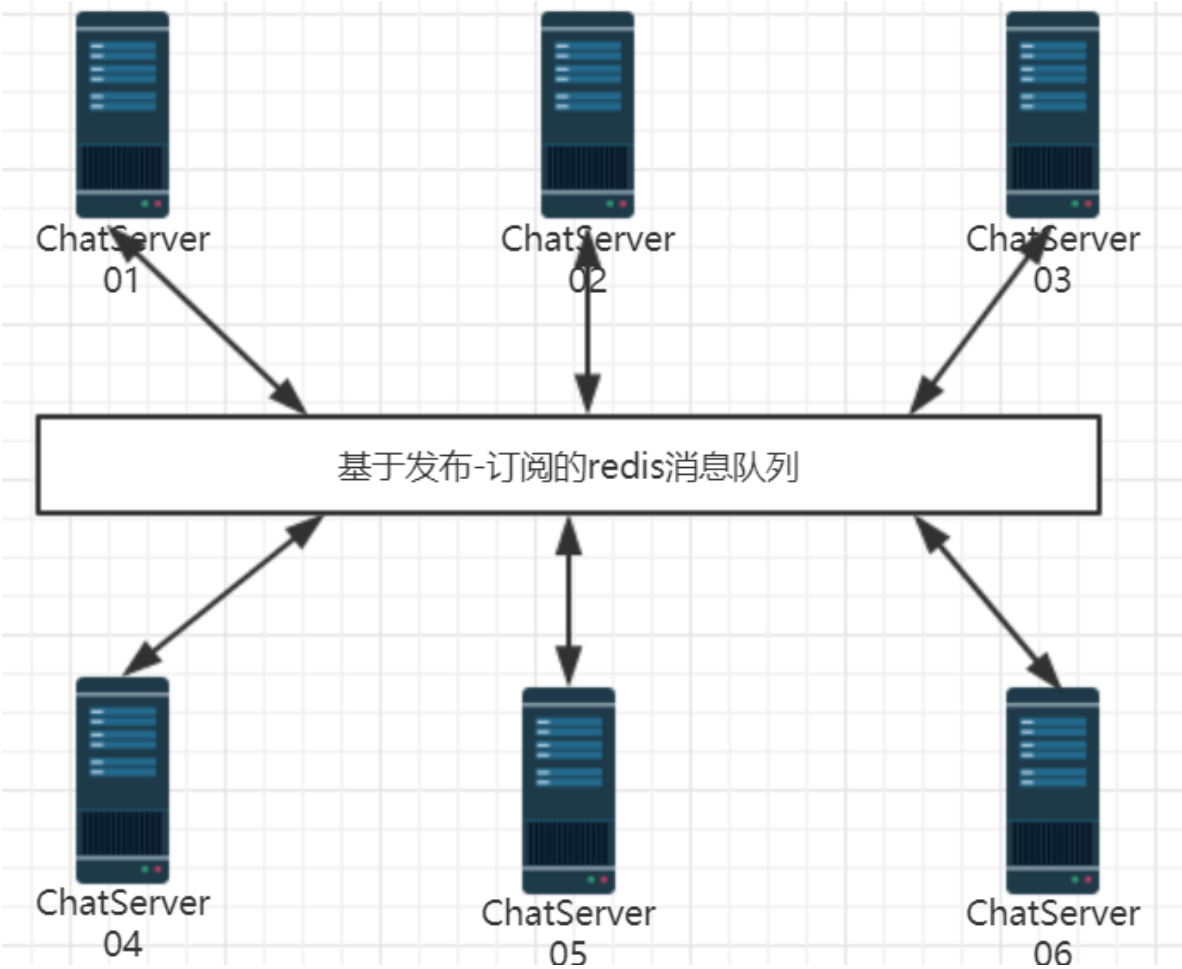
集群服务器之间的通信设计

当ChatServer集群部署多台服务器以后，当登录在不同服务器上的用户进行通信时，该怎么设计！如下设计好吗？



上面的设计，让各个ChatServer服务器互相之间直接建立TCP连接进行通信，相当于在服务器网络之间进行广播。这样的设计使得各个服务器之间耦合度太高，不利于系统扩展，并且会占用系统大量的socket资源，各服务器之间的带宽压力很大，不能够节省资源给更多的客户端提供服务，因此绝对不是一个好的设计。

集群部署的服务器之间进行通信，最好的方式就是引入中间件消息队列，解耦各个服务器，使整个系统松耦合，提高服务器的响应能力，节省服务器的带宽资源，如下图所示：



在集群分布式环境中，经常使用的中间件消息队列有ActiveMQ、RabbitMQ、Kafka等，都是应用场景广泛并且性能很好的消息队列，供集群服务器之间，分布式服务之间进行消息通信。限于我们的项目业务类型并不是非常复杂，对并发请求量也没有太高的要求，因此我们的中间件消息队列选型的是-基于发布-订阅模式的redis。

redis环境安装和配置

```
tony@tony-virtual-machine:~$ sudo apt-get install redis-server    # ubuntu命令安装redis服务
```

ubuntu通过上面命令安装完redis，会自动启动redis服务，通过ps命令确认：

```
tony@tony-virtual-machine:~$ ps -ef | grep redis
redis       2717      1  0 13:24 ?        00:00:00 /usr/bin/redis-server 127.0.0.1:6379
```

可以看到redis默认工作在本地主机的6379端口上。

redis发布-订阅相关命令

redis首先是一个强大的缓存服务器，比memcache强大很多，不仅仅支持多种数据结构（不像memcache只能存储字符串）如字符串、list列表、set集合、map映射表等结构，还可以支持数据的持久化存储（memcache只支持内存存储），经常被应用到高并发的服务器环境设计之中。

启动redis-cli客户端，连接redis server体验一下数据缓存功能，如下：

redis存储普通key-value：

```
tony@tony-virtual-machine:~$ redis-cli
127.0.0.1:6379> set 1 "zhang san"  # 设置key-value
OK
127.0.0.1:6379> get 1
"zhang san"
127.0.0.1:6379> set num 1
OK
127.0.0.1:6379> INCR num   # redis本身支持事务处理，多线程对key自增自减是线程安全的
(integer) 2
127.0.0.1:6379> INCR num
(integer) 3
```

redis列表操作：

```
127.0.0.1:6379> lpush arr a
(integer) 1
127.0.0.1:6379> lpush arr b
(integer) 2
127.0.0.1:6379> lpush arr c
(integer) 3
127.0.0.1:6379> lpush arr d
(integer) 4
127.0.0.1:6379> LRANGE 0 -1
(error) ERR wrong number of arguments for 'lrange' command
127.0.0.1:6379> LRANGE arr 0 -1
1) "d"
2) "c"
3) "b"
4) "a"
127.0.0.1:6379> RPOP arr
"a"
127.0.0.1:6379> LRANGE arr 0 -1
1) "d"
2) "c"
3) "b"
```

redis甚至还可以支持阻塞式的key读取，用BRPOP命令，如下

```
127.0.0.1:6379> BRPOP arr 0
1) "arr"
2) "b"
127.0.0.1:6379> BRPOP arr 0
1) "arr"
2) "c"
127.0.0.1:6379> BRPOP arr 0
1) "arr"
2) "d"
127.0.0.1:6379> BRPOP arr 0
 # 此处由于arr列表已经没有元素，因此读取阻塞了
```

redis对于更复杂的类型操作，大家可以自行查阅资料。

redis的发布-订阅机制：发布-订阅模式包含了两种角色，分别是**消息的发布者**和**消息的订阅者**。订阅者可以订阅一个或者多个频道channel，发布者可以向指定的频道channel发送消息，所有订阅此频道的订阅者都会收到此消息。

订阅频道的命令是 subscribe，可以同时订阅多个频道，用法是 subscribe channel1 [channel2 ...]，如下：

```
tony@tony-virtual-machine:~$ redis-cli
127.0.0.1:6379> SUBSCRIBE "zhang san"
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "zhang san"
3) (integer) 1
               # <============ 此处订阅了"zhang san"这个频道，进入订阅阻塞状态，等待该频道上的信息
```

执行上面命令客户端会进入订阅状态，处于此状态下客户端不能使用除subscribe、unsubscribe、psubscribe和punsubscribe这四个属于"发布/订阅"之外的命令，否则会报错。

打开另一个redis-cli客户端，给"zhang san"频道发布消息，如下：

```
tony@tony-virtual-machine:~$ redis-cli
127.0.0.1:6379> publish "zhang san" "hello world!"
(integer) 1
```

第一个redis-cli客户端接收到"zhang san"频道的消息，如下：

```
127.0.0.1:6379> SUBSCRIBE "zhang san"
Reading messages... (press Ctrl-C to quit)
1) "subscribe"
2) "zhang san"
3) (integer) 1
1) "message"
2) "zhang san"
3) "hello world!"
```

进入订阅状态后客户端可能收到3种类型的回复。每种类型的回复都包含3个值，第一个值是消息的类型，根据消类型的不同，第二个和第三个参数的含义可能不同。消息类型的取值可能是以下3个：

1. subscribe:表示订阅成功的反馈信息。第二个值是订阅成功的频道名称，第三个是当前客户端订阅的频道数量。
2. message:表示接收到的消息，第二个值表示产生消息的频道名称，第三个值是消息的内容。
3. unsubscribe:表示成功取消订阅某个频道。第二个值是对应的频道名称，第三个值是当前客户端订阅的频道数量，当此值为0时客户端会退出订阅状态，之后就可以执行其他非"发布/订阅"模式的命令了。

redis发布-订阅的客户端编程

redis支持多种不同的客户端编程语言，例如Java对应jedis、php对应phpredis、C++对应的则是hiredis。下面是安装hiredis的步骤：

1. git clone https://github.com/redis/hiredis　　从github上下载hiredis客户端，进行源码编译安装

```
tony@tony-virtual-machine:~/github$ git clone
https://github.com/redis/hiredis
正克隆到 'hiredis'...
remote: Enumerating objects: 3261, done.
^C收对象中:  83% (2707/3261), 876.01 KiB | 59.00 KiB/s
```

2. cd hiredis
3. make

```
tony@tony-virtual-machine:~/github/hiredis$ make
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb net.c
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb hiredis.c
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb sds.c
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb async.c
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb read.c
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb sockcompat.c
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb sslio.c
cc -shared -Wl,-soname,libhiredis.so.0.14 -o libhiredis.so net.o hiredis.o
sds.o async.o read.o sockcompat.o sslio.o
ar rcs libhiredis.a net.o hiredis.o sds.o async.o read.o sockcompat.o
sslio.o
cc -std=c99 -pedantic -c -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-
strings -Wno-missing-field-initializers -g -ggdb test.c
cc -O3 -fPIC   -Wall -W -Wstrict-prototypes -Wwrite-strings -Wno-missing-
field-initializers -g -ggdb -o hiredis-test test.o libhiredis.a
Generating hiredis.pc for pkgconfig...
tony@tony-virtual-machine:~/github/hiredis$
```

编译成功！

4. sudo make install

```
tony@tony-virtual-machine:~/github/hiredis$ sudo make install
[sudo] tony 的密码：
mkdir -p /usr/local/include/hiredis /usr/local/include/hiredis/adapters
/usr/local/lib
cp -pPR hiredis.h async.h read.h sds.h sslio.h /usr/local/include/hiredis
cp -pPR adapters/*.h /usr/local/include/hiredis/adapters
cp -pPR libhiredis.so /usr/local/lib/libhiredis.so.0.14
cd /usr/local/lib && ln -sf libhiredis.so.0.14 libhiredis.so
cp -pPR libhiredis.a /usr/local/lib
mkdir -p /usr/local/lib/pkgconfig
cp -pPR hiredis.pc /usr/local/lib/pkgconfig
```

拷贝生成的动态库到/usr/local/lib目录下！

5. sudo ldconfig /usr/local/lib

如何通过C++使用hiredis客户端进行subscribe 和publish编程，将在课堂上面做详细讲解。

数据库设计

表设计

User表

字段名称	字段类型	字段说明	约束
id	INT	用户id	PRIMARY KEY、AUTO_INCREMENT
name	VARCHAR(50)	用户名	NOT NULL, UNIQUE
password	VARCHAR(50)	用户密码	NOT NULL
state	ENUM('online', 'offline')	当前登录状态	DEFAULT 'offline'

Friend表

字段名称	字段类型	字段说明	约束
userid	INT	用户id	NOT NULL、联合主键
friendid	INT	好友id	NOT NULL、联合主键

AllGroup表

字段名称	字段类型	字段说明	约束
id	INT	组id	PRIMARY KEY、AUTO_INCREMENT
groupname	VARCHAR(50)	组名称	NOT NULL,UNIQUE
groupdesc	VARCHAR(200)	组功能描述	DEFAULT ''

GroupUser表

字段名称	字段类型	字段说明	约束
groupid	INT	组id	NOT NULL、联合主键
userid	INT	组员id	NOT NULL、联合主键
grouprole	ENUM('creator', 'normal')	组内角色	DEFAULT ‘normal’

OfflineMessage表

字段名称	字段类型	字段说明	约束
userid	INT	用户id	NOT NULL
message	VARCHAR(500)	离线消息（存储Json字符串）	NOT NULL

MySQL数据库环境搭建和编程

MySQL环境安装设置

ubuntu环境安装mysql-server和mysql开发包，包括mysql头文件和动态库文件，命令如下：

```
sudo apt-get install mysql-server     =》 安装最新版MySQL服务器
sudo apt-get install libmysqlclient-dev =》 安装开发包
```

ubuntu默认安装最新的mysql，但是初始的用户名和密码是自动生成的，按下面步骤修改mysql的root用户密码为123456

```
【step 1】tony@tony-virtual-machine:~$ sudo cat /etc/mysql/debian.cnf

[client]
host     = localhost
user     = debian-sys-maint         《==============  初始的用户名
password = Kk3TbShbFNvjvhpM       《================ 初始的密码
socket   = /var/run/mysqld/mysqld.sock

【step 2】用上面初始的用户名和密码，登录mysql server，修改root用户的密码，命令如下：

tony@tony-virtual-machine:~$ mysql -u debian-sys-maint -pKk3TbShbFNvjvhpM


命令解释： -u后面是上面查看的用户名 -p后面紧跟上面查看的密码

mysql> update mysql.user set authentication_string=password('123456') where
user='root' and host='localhost';

mysql> update mysql.user set plugin="mysql_native_password";

mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)

mysql> exit
Bye
```

【step 3】重新用root和123456登录mysql-server

```
tony@tony-virtual-machine:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
Your MySQL connection id is 9
Server version: 5.7.26-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

【step 4】设置MySQL字符编码utf-8，可以支持中文操作

```
mysql> show variables like "char%";    # 先查看MySQL默认的字符编码
+--------------------------+----------------------------+
| Variable_name            | Value                      |
+--------------------------+----------------------------+
| character_set_client     | utf8                       |
| character_set_connection | utf8                       |
| character_set_database   | latin1                     |
| character_set_filesystem | binary                     |
| character_set_results    | utf8                       |
| character_set_server     | latin1                     |    《============不支持中
文！！！
| character_set_system     | utf8                       |
| character_sets_dir       | /usr/share/mysql/charsets/ |
+--------------------------+----------------------------+
8 rows in set (0.06 sec)

mysql> set character_set_server=utf8;
Query OK, 0 rows affected (0.00 sec)
```

修改表的字符编码：alter table user default character set utf8;

修改属性的字符编码：alter table user modify column name varchar(50) character set utf8;

MySQL数据库编程

```
// 数据库配置信息
static string server = "127.0.0.1";
static string user = "root";
static string password = "123456";
static string dbname = "chat";

// 数据库操作类
class MySQL
{
public:
    // 初始化数据库连接
    MySQL()
    {
        _conn = mysql_init(nullptr);
    }
    // 释放数据库连接资源
```

```
    ~MySQL()
    {
        if (_conn != nullptr)
            mysql_close(_conn);
    }
    // 连接数据库
    bool connect()
    {
         MYSQL *p = mysql_real_connect(_conn, server.c_str(), user.c_str(),
            password.c_str(), dbname.c_str(), 3306, nullptr, 0);
         if (p != nullptr)
         {
             mysql_query(_conn, "set names gbk");
         }
         return p;
    }
    // 更新操作
    bool update(string sql)
    {
        if (mysql_query(_conn, sql.c_str()))
        {
            LOG_INFO << __FILE__ << ":" << __LINE__ << ":"
                << sql << "更新失败!";
            return false;
        }
        return true;
    }
    // 查询操作
    MYSQL_RES* query(string sql)
    {
        if (mysql_query(_conn, sql.c_str()))
        {
            LOG_INFO << __FILE__ << ":" << __LINE__ << ":"
                << sql << "查询失败!";
            return nullptr;
        }
        return mysql_use_result(_conn);
    }
private:
    MYSQL *_conn;
};
```

这里用UserModel示例，通过UserModel如何对业务层封装底层数据库的操作。代码示例如下：

```
class UserModel
{
public:
    // 重写add接口方法，实现增加用户操作
    bool add(UserDO &user)
    {
        // 组织sql语句
        char sql[1024] = { 0 };
        sprintf(sql, "insert into user(name,password,state) values('%s', '%s',
'%s')",
            user.getName().c_str(),
            user.getPwd().c_str(),
            user.getState().c_str());
```

```
        MySQL mysql;
        if (mysql.connect())
        {
            if (mysql.update(sql))
            {
                LOG_INFO << "add User success => sql:" << sql;
                return true;
            }
        }
        LOG_INFO << "add User error => sql:" << sql;
        return false;
};
```