天翼云分布式消息服务

RocketMQAPI 说明文档（C++）

中国电信股份有限公司云计算分公司

目　录

1. 概述

CtgMQ C++客户端支持基本的消息发送、消费、查询功能。包括：

1) 同步、异步发送

2) Pull 方式拉取消费

3) 消息签收

4) 消息内容、消费状态查询

2. 接口详细说明

2.1 查询接口

2.1.1 根据查询条件从消息中间件中查询消息

bool QueryMessage(const QueryCondition& query_condition,
QueryResult* result)

功能描述：根据条件查询消息

入参说明：

参数	类型	是否可为空	说明
query_condition	QueryCondition	N	查询条件；详见 QueryCondition 定义（query_defines.h）

输出说明：

参数	类型	说明

result	QueryResult	查询结果，详见 QueryResult 定义，其中 consume_status 表示消费状态

参数	类型	说明
函数返回值	bool	是否启动成功

2.2消息者生产者

2.2.1 创建消息生产者

Producer* CreateProducer(const std::string& producer_group);

功能描述：创建消息生产者

只指定消息生产者所在的组，其他生产者配置为默认值

入参说明：

参数	类型	是否可为空	说明
producer_group	string	N	生产者组的名称

输出说明：

参数	类型	说明
函数返回值	Producer*	消息生产者

Producer* CreateProducer(const ProducerConfig& config);

功能描述：创建消息生产者

根据生产者配置信息创建一个消息生产者

入参说明：

参数	类型	是否可为空	说明
config	ProducerConfig	N	生产者配置信息

输出说明：

参数	类型	说明
函数返回值	Producer*	消息生产者

注意：创建生产者的 config 中 namesrv 的登录密码需要是 md5 加密（md5 32 位小写），例如：

密码为 123456，则这样设置：

```
config.MutableClientConfig()->SetAuthPWD("e10adc3949ba59abbe56e057f20f883e");
```

2.2.2 启动消息生产者

bool Start() ;

功能描述：启动消息生产者服务

入参说明：

无

输出说明：

参数	类型	说明
函数返回值	bool	是否启动成功

2.2.3 关闭消息生产者

void Shutdown()

功能描述：关闭服务，一旦关闭，此对象将不可用。

入参说明：

无

输出说明：

无

2.2.4 同步方式发送消息

SendResult Send(const Message& msg)；

功能描述：同步发送消息

入参说明：

参数	类型	是否可为空	说明
msg	Message	N	消息对象

输出说明：

参数	类型	说明
函数返回值	SendResult	发送消息的返回结果

2.2.5 异步方式发送消息

void Send(const Message& msg, SendCallback* done)；

功能描述：异步方式发送消息

入参说明：

参数	类型	是否可为空	说明
msg	Message	N	消息对象
done	SendCallback	N	发送完消息后的调用的回调函数

输出说明：

参数	类型	说明
函数返回值	Void	无返回值

2.2.6 获取生产者配置

ProducerConfig& GetConfig();

功能描述：获取生产者配置信息

入参说明：

无

输出说明：

参数	类型	说明
函数返回值	ProducerConfig	生产者配置信息，详情见生产者配置说明

2.2.7 同步发送到指定队列

SendResult send(const Message& msg,const MessageQueue& mq);

功能描述：同步发送消息到指定队列

入参说明：

参数	类型	是否可为空	说明

msg	Message	N	消息对象
mq	MessageQueue	N	指定队列

输出说明：

参数	类型	说明
函数返回值	sendResult	发送结果

2.3消息消费者（V1）

V1 消费提供按队列和按主题两种消费方式：

1. 按队列消费（默认）：先获取主题对应所有队列，拉取消息，消息消费完成后，自行调用接口更新消费进度到服务端；
2. 按主题消费：

（1）按主题消费在创建消费者时，v1_consume_method 配置需要设置为 CONSUME_BY_TOPIC；

（2）按主题消费由 sdk 实现队列的负载均衡，并定时更新消费进度到服务端，应用无须再更新消费进度；

（3）按主题消费仅支持无序消费。

注意事项：默认是按队列消费，应用只能选择其中一种消费方式进行消费，不支持混合使用。

2.3.1 创建消息消费者

PullConsumerV1* CreateConsumerV1 (const std::string& consumer_group);

功能描述：创建 V1 消息消费者

只指定消息消费者所在的组，其他消费者配置为默认值

入参说明：

参数	类型	是否可为空	说明
consumer _group	string	N	消费者组的名称

输出说明：

参数	类型	说明
函数返回值	PullConsumer *	消息消费者

PullConsumerV1* CreateConsumerV1 (const ConsumerConfig& config);

功能描述：创建 V1 消息消费者（原生）

根据消费者配置信息创建一个 V1 消息消费者

入参说明：

参数	类型	是否可为空	说明
config	ConsumerConfig	N	消费者配置信息

输出说明：

参数	类型	说明
函数返回值	PullConsumer *	消息消费者

注意：创建消费者的 config 中 namesrv 的登录密码需要是 md5 加密（md5 32 位小写），例如：

密码为 123456，则这样设置：

```
config.MutableClientConfig()->SetAuthPWD("e10adc3949ba59abbe56e057f20f883e");
```

2.3.2 启动消息消费者

bool Start() ;

功能描述：启动 V1 消息消费者服务

入参说明：

无

输出说明：

参数	类型	说明
函数返回值	bool	是否启动成功

2.3.3 关闭消息消费者

void Shutdown()

功能描述：关闭服务，一旦关闭，此对象将不可用。

入参说明：

无

输出说明：

无

2.3.4 V1 消费（按队列）

2.3.4.1 查询消费者在指定队列上的消费进度

int64_t QueryConsumeProgress(const MessageQueue& mq) = 0;

功能描述：查询消费者在指定队列上的消费进度

入参说明：

参数	类型	是否可为空	说明
mq	MessageQueue	N	指定队列

输出说明：

参数	类型	说明
函数返回值	int64_t	所查询队列的消费进度

2.3.4.2 同步从指定队列的指定位置拉取消息

```
bool Consume(const MessageQueue& mq,
             const std::string& sub_expression,
             int64_t max_num,
             int64_t timeout_in_ms,
             int64_t startOffset,
             PullResult* result) = 0;
```

功能描述：同步从指定队列的指定位置拉取消息

入参说明：

参数	类型	是否可为空	说明
mq	MessageQueue	N	所指定的队列
sub_expression	string	N	订阅过滤表达式，支持与运算 eg "tag1 || tag2 || tag3" "" 和 "*" 表示全部订阅
max_num	int64_t	N	最大消息条数（一次最大拉取 32 条）

timeout_in_ms	int64_t	N	超时时间（单位：毫秒）
startOffset	int64_t	N	指定位置

输出说明：

参数	类型	说明
result	PullResult*	返回数据结构，如果消费成功，里面存储了消息的消息列表

参数	类型	说明
函数返回值	bool	是否消费成功

2.3.4.3 更新指定队列的消费进度

void UpdateConsumeOffset(const MessageQueue& mq, int64_t offset) = 0;

功能描述：更新指定队列的消费进度

入参说明：

参数	类型	是否可为空	说明
mq	MessageQueue	N	指定队列
offset	int64_t	N	需要更新的消费进度偏移值

输出说明：

参数	类型	说明
函数返回值	void	是否更新成功

2.3.5 V1 消费（按主题）

2.3.5.1 消费消息

bool ConsumeByTopic(const std::string& topic,

int64_t max_num,

int64_t timeout_in_ms,

PullResult* result) = 0;

功能描述：主题消费，根据 TOPIC 从消息中间件中消费消息（无序）

入参说明：

参数	类型	是否可为空	说明
topic	string	N	主题名字
max_num	int64_t	N	最大消息条数（一次最大拉取 32 条）
timeout_in_ms	int64_t	N	超时时间（单位：毫秒）

输出说明：

参数	类型	说明
result	PullResult*	返回数据结构，如果消费成功，里面存储了消息的消息列表

参数	类型	说明
函数返回值	bool	是否消费成功

2.3.5.2 根据过滤条件进行消费

bool ConsumeByTopic (const std::string& topic,

const std::string& sub_expression,

int64_t max_num,

int64_t timeout_in_ms,

PullResult* result) = 0;

功能描述：根据过滤条件进行主题消费

入参说明：

参数	类型	是否可为空	说明
topic	string	N	主题名字
sub_expression	string	N	订阅过滤表达式，支持与运算 eg "tag1 || tag2 || tag3" "" 和 "*" 表示全部订阅
max_num	int64_t	N	最大消息条数
timeout_in_ms	int64_t	N	超时时间（单位：毫秒）

输出说明：

参数	类型	说明
result	PullResult*	返回数据结构，如果消费成功，里面存储了消息的消息列表

参数	类型	说明
函数返回值	bool	是否消费成功

2.3.5.3消息签收

AckResult AckMessage (const MessageExt& msg ,

ConsumeStatus status = CONSUME_SUCCESS) = 0;

功能描述：消息签收

入参说明：

参数	类型	是否可为空	说明
msg	MessageExt	N	签收的消息对象
status	ConsumeStatus	Y	CONSUME_SUCCESS：签收成功 RECONSUME_LATER：签收失败

输出说明：

参数	类型	说明
函数返回值	AckResult	消息签收结果 包含错误码，和错误信息描述

2.3.5.4 消费重试队列的消息

bool ConsumeSendBackMessage(int64_t max_num,

int64_t timeout_in_ms,

PullResult* result) = 0;

功能描述：消费重试队列的消息（签收失败的消息将进入重试队列，需调该接口进行消费重试队列的消息）。

入参说明：

参数	类型	是否可为空	说明
max_num	int64_t	N	最大消息条数（一次最大拉取 32 条）

timeout_in_ms	int64_t	N	超时时间（单位：毫秒）

输出说明：

参数	类型	说明
result	PullResult*	返回数据结构，如果消费成功，里面存储了消息的消息列表

参数	类型	说明
函数返回值	bool	是否消费成功

2.3.6 获取消费者配置

ConsumerConfig & GetConfig();

功能描述：获取 V1 消费者配置信息

入参说明：

无

输出说明：

参数	类型	说明
函数返回值	ConsumerConfig	消费者配置信息，详情见消费者配置说明

2.4管理类

2.4.1 创建消息管理者

Adminer* CreateAdminer(const std::string& admin_group);

功能描述：创建消息管理者

只指定消息管理者所在的组，其他管理者配置为默认值

入参说明：

参数	类型	是否可为空	说明
admin_group	string	N	管理者组的名称

输出说明：

参数	类型	说明
函数返回值	Adminer *	消息管理者

Adminer* CreateAdminer(const AdminerConfig& config);

功能描述：创建消息管理者

根据管理者配置信息创建一个消息管理者

入参说明：

参数	类型	是否可为空	说明
config	AdminerConfig	N	管理者配置信息

输出说明：

参数	类型	说明
函数返回值	Adminer *	消息管理者

注意：创建管理者的 config 中 namesrv 的登录密码需要是 md5 加密（md5 32 位小写），例如：

密码为 123456，则这样设置：

```
config.MutableClientConfig()->SetAuthPWD("e10adc3949ba59abbe56e057f20f883e");
```

2.4.2 启动消息管理者

bool Start() ;

功能描述：启动消息管理者服务

入参说明：

无

输出说明：

参数	类型	说明
函数返回值	bool	是否启动成功

2.4.3 关闭消息管理者

void Shutdown()

功能描述：关闭服务，一旦关闭，此对象将不可用。

入参说明：

无

输出说明：

无

2.4.4 查询指定主题包含的全部队列

set<MessageQueue> fetchSubscribeMessageQueues(const std::string& topic);

功能描述：查询指定主题包含的全部队列

入参说明：

参数	类型	是否可为空	说明
topic	std::string	N	指定的主题名

输出说明：

参数	类型	说明
函数返回值	set<MessageQueue>	消息队列集

2.4.5 查询指定队列上的消息总数

long queryMsgCount(MessageQueue& mq);

功能描述：查询指定队列上的消息总数

入参说明：

参数	类型	是否可为空	说明
mq	MessageQueue	N	消息队列

输出说明：

参数	类型	说明
函数返回值	long	消息总数

2.4.6 查询指定消费者在指定队列上未消费的消息总数

long queryNoCousumeMsgCount(const std::string& consumerGroup,MessageQueue& mq);

功能描述：查询指定消费者在指定队列上未消费的消息总数

入参说明：

参数	类型	是否可为空	说明
consumerGroup	std::string	N	指定的消费者组 ID
mq	MessageQueue	N	消息队列

输出说明：

参数	类型	说明
函数返回值	long	消息总数

2.4.7 根据 topic 和 key 查询消息

QueryResult queryMessage(std::string& topic, std::string& key, int num, long begin, long end)；

功能描述：根据 topic 和 key 查询消息

入参说明：

参数	类型	是否可为空	说明
topic	std::string	N	指定的主题名
key	std：：string	N	key 值
num	int	N	返回的消息条数
begin	long	N	开始时间戳
end	long	N	结束时间戳

输出说明：

参数	类型	说明
函数返回值	QueryResult	查询结果

2.4.8 查询指定队列上的最小偏移量

long queryMinOffset(MessageQueue& mq);

功能描述：查询指定队列上的最小偏移量

入参说明：

参数	类型	是否可为空	说明
mq	MessageQueue	N	消息队列

输出说明：

参数	类型	说明
函数返回值	long	最小偏移量

2.4.9 查询指定队列上的最大偏移量

long queryMaxOffset(MessageQueue& mq);

功能描述：查询指定队列上的最大偏移量

入参说明：

参数	类型	是否可为空	说明
mq	MessageQueue	N	消息队列

输出说明：

参数	类型	说明
函数返回值	long	最大偏移量

3. 配置文件说明

3.1 客户端配置

生产者消费者中都有的配置

```
/**
 * 客户端配置类，包含客户端相关的配置
 * Producer 和 Consumer 各自配置都包含该配置，根据 instance name，创建不同的内部 Client 对象
 * 各项配置说明如下:
 *    ----------client configure----------
 *    - std::string namesrv_addr;
 *      配置服务器地址，默认从'NAMESRV_ADDR'环境变量获取
 *    - std::string auth_id;
 *      鉴权账号
 *    - std::string auth_pwd;
 *      鉴权密码（md5 32 位加密）
 *    - std::string tenantID;
 *          租户 id
 *    - std::string clusterName
 *      broker 集群名
 *    - std::string client_ip;
```

* 客户端 IP，默认内网第一个 IP
* - std::string instance_name;
* 实例名，默认为进程 pid
* - int64_t poll_namesrv_interval_in_ms;
* 从配置服务器更新配置时间间隔，默认为 30s
* - int64_t heartbeat_broker_interval_in_ms;
* 心跳 broker 间隔，默认为 30s
* - int64_t persist_consumer_offset_interval_in_ms;
* 持久化消费进度间隔，默认 5000ms
* - bool wait_all_pending_request_finish;
* 是否等待所有请求完成再退出，默认 false
* - std::string log_dir;
* 日志目录，默认为/tmp
* - int32_t log_max_size;
* 日志文件最大大小，MB 为单位,默认情况下没有大小限制
* - int32_t min_log_level; (INFO 0, WARNING 1, ERROR 2, FATAL 3)
* 最低输出日志级别，默认为 0, 低于该级别的日志不输出
* - int32_t log_verbose_level;
* verbose 日志级别，默认为 0
* - MQAlarmStrategy alarm_strategy;
* 告警策略。默认是将告警信息写入本地文件中，接入告警平台的选项暂时还没有实现
* ----------network configure----------
* - int32_t work_thread_num

```
*      网络工作线程数目，默认为 3，如网络请求频繁，提高线程数
*    - int32_t callback_thread_num
*      网络回调处理线程数目，默认为 4, 如异步调用回调逻辑重，提高线程数
*    - bool vipChannelEnabled
*      是否启动发送消费分离，默认为 true，发送端口为 broker 端口-2；如果为
false，则发送访问 broker 默认端口
*/
```

3.2生产者配置

```
/**
* Producer 配置类，包含 Producer 相关的配置
* 各项配置说明如下:
*    - ClientConfig client_config;
*      客户端配置，见 ClientConfig 注释
*    - std::string producer_group;
*      生产者组名，主要用于事务消息回查与统计监控，默认
"DEFAULT_PRODUCER", 必须设置
*    - int64_t send_message_timeout_in_ms;
*      发送消息超时时间，默认 6000ms
*    - int64_t max_message_size;
*      最大消息大小，默认 128k
*    - uint32_t max_retry_times;
*      发送消息失败后的重试次数，默认值为 0 不重试(只有指定几种网络失败
的场景需要重试)
*    - bool retry_failed_lock_mq
```

* 当发送结果为 FLUSH_DISK_TIMEOUT, FLUSH_SLAVE_TIMEOUT, SLAVE_NOT_AVALIABLE 时，设置相应 broker 上的 topic 权限为不可写

注：如果客户端不重启，则 rocketmq 的 topic 权限更改为延迟在客户端生效，需要等待 topic 路由信息更新。

* （**注意：此配置是在需要非常严格的数据安全性的场景使用，开启自动禁写，一旦出现如上说明的异常，将会直接禁止 topic 写入，客户端无法再次发送消息到此 topic，直到人工介入恢复。强烈建议不要开启。**）

* int64_t compressMsgBodyOverHowmuch

* 消息体多大时开始压缩，默认 4k

*/

3. 3消费者配置

```
/**
 * Consumer 配置类，包含 Consumer 相关的配置
 * 各项配置说明如下:
 *  - ClientConfig client_config;
 *    客户端配置，见 ClientConfig 注释
 *  - std::string consumer_group;
 *    消费者组名，有多处重要作用，默认"DEFAULT_CONSUMER", 必须设置
 *  - bool consume_orderly;
 *    是否顺序消费，顺序消费只能消费签收完，才能消费下一次，默认 false（可并发消费）
 *  - int64_t ack_timeout_in_ms;
```

```
*      发送 ack 给服务端的超时处理时间。小于等于 0 时不做超时处理.默认值
0，不做 ack 超时处理
*    - MsgTimeoutStrategy timeout_strategy;
*      发送 ack 超时的超时处理策略，默认策略是 ALARM
*    - V1_CONSUME_METHOD v1_consume_method;
*    消费方式:1.CONSUME_BY_QUEUE:按队列消费(默
认);2.CONSUME_BY_TOPIC:按主题消费
*/
```

3. 4管理者配置

```
/**
* Adminer 配置类，包含 Adminer 相关的配置
* 各项配置说明如下:
*    - ClientConfig client_config;
*      客户端配置，见 ClientConfig 注释
*    - std::string adminer_group;
*      管理者组名
*/
```

4. 错误码说明

发送消息错误码

SendResult::ResultCode 错误码分成‘明确错误’与‘不确定错误’两类，不确定错误是指发送后，不确定是否发送成功，比如发送请求超时，发生这类错误，重发

可能导致重复，因此需要应用通过查看错误信息、记录日志、按消息 key 查询消息、查询业务逻辑等方式介入处理：

（1） 以下为明确失败状态

错误码	Int 值	说明
SEND_FAIL	4	发送失败
CLIENT_SERVICE_NOT_OK	6	客户端不可用
MESSAGE_ILLEGAL	7	消息不合法
PROCESS_RESPONSE_ERROR	8	处理应答错误
SERVICE_NOT_AVAILABLE	9	服务不可用，可能是正在关闭或者权限问题, 或磁盘空间满了
NO_TOPIC_ROUTE_INFO	10	TOPIC 的路由不存在
TOPIC_NOT_EXIST,	11	Broker, Topic 不存在
NO_PERMISSION	13	Broker, Namesrv 无权限执行此操作，可能是发、收、或者其他操作

（2） 以下为异常不确定状态(对于发送消息，重试可能导致重复)

错误码	int 值	说明
FLUSH_DISK_TIMEOUT	1	Broker 刷盘超时
FLUSH_SLAVE_TIMEOUT	2	Broker 同步双写，等待 Slave 应答超时
SLAVE_NOT_AVAILABLE	3	Broker 同步双写，Slave 不可用
REMOTING_ERROR	5	网络故障，如网络超时
SYSTEM_ERROR	12	系统错误，如服务端逻辑异常

UNKNOW_ERROR	14	未知错误，查看错误信息

消费消息错误码说明

错误码	int 值	说明
NO_MESSAGE	1	没有新消息可消费
ERROR	2	发生错误，错误信息请打印 error_msg
CLIENT_SERVICE_NOT_OK	3	客户端状态异常，一般是没有调用 Start，或者已经 Shutdown
PREVIOUS_MESSAGE_IS_CONSUMING	4	消息签收相关，顺序消息需要等上一批消息签收完，才能继续消费
PROCESS_QUEUE_NOT_ACTIVE	5	集群消息相关，当前 topic 队列没有激活，可能仍被同一个 consumer group 的其他 consumer 实例占用
PULL_TIMEOUT	6	拉取超时，一般需要重试
TOO_MUCH_MESSAGE_NOT_ACK	7	太多消息没有签收，不能拉取新消息，默认每个 topic 最大未签收数为 5w
FAIL_TO_FETCH_MESSAGE_QUEUE	8	获取 topic 队列信息失败，可能 1. topic 没有创建 2. 同一个消费组启动的实例数大于 topic 的队列数
OFFSET_ILLEGAL	9	非法的 offset，可能过大或过小
TOPIC_NOT_EXIST	17	主题不存在

SUBSCRIPTION_GROUP_NOT_EXIST	26	订阅组不存在
SUBSCRIPTION_NO_PERMISSION	27	订阅组没消费权限
CONSUME_GROUP_MECHANISM_ERROR	518	订阅组消费机制有误，使用的消费类跟新建订阅组时指定的消费机制不同
CONSUME_GROUP_MECHANISM_NOT_EXIST	519	订阅组没有指定消费机制，使用的是原生消费机制，还是 BDB 消费机制
TOPIC_DISALLOWD_CONSUMER_GROUP	621	订阅组没权限消费此主题
REQUEST_QUEUEID_ILLAGAL	624	请求消费的 queueId 错误，检查队列有效性失败

签收\更新消息错误码说明

错误码	int 值	说明
MESSAGE_ALREADY_ACKED	1	消息已经被签收，不能重复签收
MESSAGE_IS_NOT_CONSUMING	2	消息没有在消费，不能签收
SKIP_ACK_NOT_ALLOWED_FOR_ORDER_MESSAGE	3	顺序消费不能跳跃签收
MESSAGE_ID_INVALID	4	消息 ID 非法，不能按 id 签收
REMOTING_ERROR	5	网络错误
SYSTEM_ERROR	6	系统错误
INVALID_ARGUMENT	7	参数非法

5. 应用设计建议

为了提高 V1 按队列消费性能，建议参考以下方式进行设计：

1. 单次拉取 32 条消息；

2. 多次拉取消息（如：拉取 30 次），再进行批量更新一次消费进度；

3. 单次拉取消息后，判断距离上一次更新进度时间如果超过 5 秒钟，就更新一次消费进度；

4. 起多个线程，分别消费不同的队列。

用户在实际的应用设计时，为了获得较高的性能，可采用多线程并行处理的方式来进行消息处理。这里给出两种设计建议方式：

方式一： 起一个查询线程，调用 fetchSubscribeMessageQueues 接口负责查询该主题下对应的所有消息队列，假设主题对应有 20 个队列，起 10 个消息处理线程，平均每个线程完成 2 个队列的消息消费和消息的应用逻辑处理；

注意：须保证同一个队列只能由一个线程去消费消息，否则多个线程消费同一个队列可能导致 offset 错乱。

方式二：在消费消息时，可由一个消息消费线程定时消费消息，先预拉消息并缓存到本地，然后通过多个消息处理线程来从本地获取消息，然后完成消息的应用逻辑处理。

6. 编译

操作系统为 Centos7.2，gcc 4.8.5，将 libmq_client.a 放在 lib 目录下，mq_client 的 include 头文件目录放在当前目录下。如 SyncSend.cpp 文件，编译命令参考如下：

g++ SyncSend.cpp -o SyncSend -L./lib -I. -lmq_client -lpthread -lz

7. 日志

用户可以根据应用实际情况在 config 里设置日志路径和级别（日志级别建议设置为 2，即 ERROR 级别，方便后续问题分析）。

示例：

config.MutableClientConfig()->SetLogDir("/tmp/log");//绝对路径，用户根据实际情况指定

config.MutableClientConfig()->SetMinLogLevel(2);//INFO 0, WARNING 1, ERROR 2, FATAL 3

8. 第三方库版本信息

CtgMQ C++客户端使用了一些第三方库（这里列出第三方库版本信息，编译时不需指定），包括：

1. protobuf-2.5.0
2. boost-1.53.0
3. rapidjson-1.0
4. jsoncpp-0.6.0