Redis的发布与订阅功能，由SUBSCRIBE，PSUBSCRIBE，UNSUBSCRIBE，PUNSUBSCRIBE，以及PUBLISH等命令实现。

通过执行SUBSCRIBE命令，客户端可以订阅一个或多个频道。当有客户端通过PUBLISH命令向某个频道发布消息时，频道的所有订阅者都会收到这条消息。

除了订阅具体的频道之外，客户端还可以通过执行PSUBSCRIBE命令订阅一个或多个频道模式。当有客户端通过PUBLISH命令向某个频道发布消息时，消息不仅会被发送给这个频道的所有订阅者，它还会发送给所有与这个频道相匹配的频道模式的订阅者。

UNSUBSCRIBE和PUNSUBSCRIBE命令，主要用于退订频道和退订频道模式。

**一：订阅**

1：数据结构

SUBSCRIBE命令主要使用字典结构。

在表示Redis服务器的redisServer结构体中，使用字典pubsub\_channels记录某个频道都由哪些客户端订阅。

struct redisServer **{**

**...**

*/\* Pubsub \*/*

dict **\***pubsub\_channels**;** */\* Map channels to list of subscribed clients \*/*

**...**

**}**

在字典pubsub\_channels中，以具体的频道名为key，而value是一个列表，该列表中记录了订阅该频道的所有客户端。

当向某频道发布消息时，就是通过查询该字典，将消息发送给订阅该频道的所有客户端。

在表示客户端的结构体redisClient中，使用字典pubsub\_channels记录该客户端都订阅了哪些频道：

**typedef** struct redisClient **{**

**...**

dict **\***pubsub\_channels**;** */\* channels a client is interested in (SUBSCRIBE) \*/*

**...**

**}** redisClient**;**

在字典pubsub\_channels中，以具体的频道名为key，而value为NULL。因此使用该字典能够快速判断客户端是否订阅了某频道。

PSUBSCRIBE命令主要使用列表结构。

在表示Redis服务器的redisServer结构体中，使用列表pubsub\_patterns记录客户端及其订阅的频道模式。列表中的元素都是pubsubPattern结构：

struct redisServer **{**

**...**

*/\* Pubsub \*/*

list **\***pubsub\_patterns**;** */\* A list of pubsub\_patterns \*/*

**...**

**};**

**typedef** struct pubsubPattern **{**

redisClient **\***client**;**

robj **\***pattern**;**

**}** pubsubPattern**;**

当向某频道发布消息时，就是通过查询该列表，将消息发送给订阅了与该频道相匹配的频道模式的所有客户端。

在表示客户端的结构体redisClient中，使用列表pubsub\_patterns记录该客户端都订阅了哪些频道模式，列表中的元素就是频道模式名：

**typedef** struct redisClient **{**

**...**

list **\***pubsub\_patterns**;** */\* patterns a client is interested in (PSUBSCRIBE) \*/*

**...**

**}** redisClient**;**

2：SUBSCRIBE命令

当客户端发来SUBSCRIBE命令之后，该命令的处理函数是subscribeCommand，代码如下：

void subscribeCommand**(**redisClient **\***c**)** **{**

int j**;**

**for** **(**j **=** 1**;** j **<** c**->**argc**;** j**++)**

pubsubSubscribeChannel**(**c**,**c**->**argv**[**j**]);**

c**->**flags **|=** REDIS\_PUBSUB**;**

**}**

代码很简单，针对命令参数中的每一个channel，调用函数pubsubSubscribeChannel，将客户端c和该channel，记录到相应数据结构中。

最后，向客户端标志位中增加REDIS\_PUBSUB标记，表示该客户端进入订阅模式；

函数pubsubSubscribeChannel的代码如下：

int pubsubSubscribeChannel**(**redisClient **\***c**,** robj **\***channel**)** **{**

dictEntry **\***de**;**

list **\***clients **=** **NULL;**

int retval **=** 0**;**

*/\* Add the channel to the client -> channels hash table \*/*

**if** **(**dictAdd**(**c**->**pubsub\_channels**,**channel**,NULL)** **==** DICT\_OK**)** **{**

retval **=** 1**;**

incrRefCount**(**channel**);**

*/\* Add the client to the channel -> list of clients hash table \*/*

de **=** dictFind**(**server**.**pubsub\_channels**,**channel**);**

**if** **(**de **==** **NULL)** **{**

clients **=** listCreate**();**

dictAdd**(**server**.**pubsub\_channels**,**channel**,**clients**);**

incrRefCount**(**channel**);**

**}** **else** **{**

clients **=** dictGetVal**(**de**);**

**}**

listAddNodeTail**(**clients**,**c**);**

**}**

*/\* Notify the client \*/*

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**subscribebulk**);**

addReplyBulk**(**c**,**channel**);**

addReplyLongLong**(**c**,**clientSubscriptionsCount**(**c**));**

**return** retval**;**

**}**

首先向字典c->pubsub\_channels中添加以该channel为key的键值对，键值对中的value为NULL。因此，该字典仅用于记录当前客户端订阅了哪些频道；

然后，在字典server.pubsub\_channels中，以channel为key寻找字典项de，如果de为NULL，则创建列表clients，以channel为key，以列表clients为value，将该键值对添加到字典server.pubsub\_channels中；如果de不为NULL，则从中取出当前已订阅该channel的客户端列表clients；然后，将客户端c添加到列表clients中；

最后，回复客户端c相应的订阅信息；

3：PSUBSCRIBE命令

当客户端发来PSUBSCRIBE命令之后，该命令的处理函数是psubscribeCommand，代码如下：

void psubscribeCommand**(**redisClient **\***c**)** **{**

int j**;**

**for** **(**j **=** 1**;** j **<** c**->**argc**;** j**++)**

pubsubSubscribePattern**(**c**,**c**->**argv**[**j**]);**

c**->**flags **|=** REDIS\_PUBSUB**;**

**}**

代码很简单，针对命令参数中的每一个频道模式，调用函数pubsubSubscribePattern，将客户端c和该频道模式，记录到相应数据结构中。

最后，向客户端标志位中增加REDIS\_PUBSUB标记，表示该客户端进入订阅模式；

函数pubsubSubscribePattern的代码如下：

int pubsubSubscribePattern**(**redisClient **\***c**,** robj **\***pattern**)** **{**

int retval **=** 0**;**

**if** **(**listSearchKey**(**c**->**pubsub\_patterns**,**pattern**)** **==** **NULL)** **{**

retval **=** 1**;**

pubsubPattern **\***pat**;**

listAddNodeTail**(**c**->**pubsub\_patterns**,**pattern**);**

incrRefCount**(**pattern**);**

pat **=** zmalloc**(sizeof(\***pat**));**

pat**->**pattern **=** getDecodedObject**(**pattern**);**

pat**->**client **=** c**;**

listAddNodeTail**(**server**.**pubsub\_patterns**,**pat**);**

**}**

*/\* Notify the client \*/*

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**psubscribebulk**);**

addReplyBulk**(**c**,**pattern**);**

addReplyLongLong**(**c**,**clientSubscriptionsCount**(**c**));**

**return** retval**;**

**}**

根据pattern，从列表c->pubsub\_patterns中寻找相应的节点，如果找不到，说明该客户端未订阅该模式，因此：首先将pattern追加到列表c->pubsub\_patterns中；然后根据c和pattern，创建pubsubPattern结构的pat，并将pat追加到列表server.pubsub\_patterns中；

最后，回复客户端相应信息；

4：订阅模式

当客户端标志位中设置了REDIS\_PUBSUB标记后，表示该客户端进入订阅模式。

在处理客户端命令的processCommand函数中，有下面的逻辑：

/\* Only allow SUBSCRIBE and UNSUBSCRIBE in the context of Pub/Sub \*/

**if** **(**c**->**flags **&** REDIS\_PUBSUB **&&**

c**->**cmd**->**proc **!=** pingCommand **&&**

c**->**cmd**->**proc **!=** subscribeCommand **&&**

c**->**cmd**->**proc **!=** unsubscribeCommand **&&**

c**->**cmd**->**proc **!=** psubscribeCommand **&&**

c**->**cmd**->**proc **!=** punsubscribeCommand**)** **{**

addReplyError**(**c**,**"only (P)SUBSCRIBE / (P)UNSUBSCRIBE / QUIT allowed in this context"**);**

**return** REDIS\_OK**;**

**}**

因此，处于订阅模式下的客户端，只能向Redis服务器发送PING、SUBSCRIBE、UNSUBSCRIBE、PSUBSCRIBE、PUNSUBSCRIBE命令。

**二：发布**

当客户端向Redis服务器发送”PUBLISH <channel> <message>”命令后，Redis服务器会将消息<message>发送给所有订阅了<channel>的客户端，以及那些订阅了与<channel>相匹配的频道模式的客户端。

PUBLISH命令的处理函数是publishCommand，代码如下：

void publishCommand**(**redisClient **\***c**)** **{**

int receivers **=** pubsubPublishMessage**(**c**->**argv**[**1**],**c**->**argv**[**2**]);**

**if** **(**server**.**cluster\_enabled**)**

clusterPropagatePublish**(**c**->**argv**[**1**],**c**->**argv**[**2**]);**

**else**

forceCommandPropagation**(**c**,**REDIS\_PROPAGATE\_REPL**);**

addReplyLongLong**(**c**,**receivers**);**

**}**

函数中，首先调用pubsubPublishMessage函数，将message发布到相应的频道；

然后，如果当前Redis处于集群模式下，则调用clusterPropagatePublish函数，向所有集群节点广播该消息；否则，调用forceCommandPropagation函数，向客户端c的标志位中增加REDIS\_FORCE\_REPL标记，以便后续能将该PUBLISH命令传递给从节点；

最后，将接收消息的客户端个数回复给客户端c；

发布消息的功能，主要是通过pubsubPublishMessage函数实现的，该函数的代码如下：

int pubsubPublishMessage**(**robj **\***channel**,** robj **\***message**)** **{**

int receivers **=** 0**;**

dictEntry **\***de**;**

listNode **\***ln**;**

listIter li**;**

*/\* Send to clients listening for that channel \*/*

de **=** dictFind**(**server**.**pubsub\_channels**,**channel**);**

**if** **(**de**)** **{**

list **\***list **=** dictGetVal**(**de**);**

listNode **\***ln**;**

listIter li**;**

listRewind**(**list**,&**li**);**

**while** **((**ln **=** listNext**(&**li**))** **!=** **NULL)** **{**

redisClient **\***c **=** ln**->**value**;**

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**messagebulk**);**

addReplyBulk**(**c**,**channel**);**

addReplyBulk**(**c**,**message**);**

receivers**++;**

**}**

**}**

*/\* Send to clients listening to matching channels \*/*

**if** **(**listLength**(**server**.**pubsub\_patterns**))** **{**

listRewind**(**server**.**pubsub\_patterns**,&**li**);**

channel **=** getDecodedObject**(**channel**);**

**while** **((**ln **=** listNext**(&**li**))** **!=** **NULL)** **{**

pubsubPattern **\***pat **=** ln**->**value**;**

**if** **(**stringmatchlen**((**char**\*)**pat**->**pattern**->**ptr**,**

sdslen**(**pat**->**pattern**->**ptr**),**

**(**char**\*)**channel**->**ptr**,**

sdslen**(**channel**->**ptr**),**0**))** **{**

addReply**(**pat**->**client**,**shared**.**mbulkhdr**[**4**]);**

addReply**(**pat**->**client**,**shared**.**pmessagebulk**);**

addReplyBulk**(**pat**->**client**,**pat**->**pattern**);**

addReplyBulk**(**pat**->**client**,**channel**);**

addReplyBulk**(**pat**->**client**,**message**);**

receivers**++;**

**}**

**}**

decrRefCount**(**channel**);**

**}**

**return** receivers**;**

**}**

代码很简单，首先根据channel，在字典server.pubsub\_channels中查找订阅了该频道的客户端列表list；针对列表中的每个客户端，向其发送message消息；

然后，轮训列表server.pubsub\_patterns，针对列表中的每一个pat，如果channel与pat->pattern相匹配，则向pat->client发送message消息；

**三：退订**

1：UNSUBSCRIBE命令

客户端发送UNSUBSCRIBE命令，用于退订之前通过SUBSCRIBE命令订阅的频道。UNSUBSCRIBE命令的处理函数是unsubscribeCommand，代码如下：

void unsubscribeCommand**(**redisClient **\***c**)** **{**

**if** **(**c**->**argc **==** 1**)** **{**

pubsubUnsubscribeAllChannels**(**c**,**1**);**

**}** **else** **{**

int j**;**

**for** **(**j **=** 1**;** j **<** c**->**argc**;** j**++)**

pubsubUnsubscribeChannel**(**c**,**c**->**argv**[**j**],**1**);**

**}**

**if** **(**clientSubscriptionsCount**(**c**)** **==** 0**)** c**->**flags **&=** **~**REDIS\_PUBSUB**;**

**}**

如果该命令没有任何参数，则调用pubsubUnsubscribeAllChannels函数，设置该客户端c退订所有频道；

否则，针对命令参数中的每一个channel，调用pubsubUnsubscribeChannel，使客户端退订相应频道；

最后，如果当前该客户端c订阅的频道和频道模式数为0，则从客户端标志位中删除REDIS\_PUBSUB标记，表示客户端退出订阅模式；

pubsubUnsubscribeAllChannels函数用于客户端退订所有频道，该函数的代码如下：

int pubsubUnsubscribeAllChannels**(**redisClient **\***c**,** int notify**)** **{**

dictIterator **\***di **=** dictGetSafeIterator**(**c**->**pubsub\_channels**);**

dictEntry **\***de**;**

int count **=** 0**;**

**while((**de **=** dictNext**(**di**))** **!=** **NULL)** **{**

robj **\***channel **=** dictGetKey**(**de**);**

count **+=** pubsubUnsubscribeChannel**(**c**,**channel**,**notify**);**

**}**

*/\* We were subscribed to nothing? Still reply to the client. \*/*

**if** **(**notify **&&** count **==** 0**)** **{**

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**unsubscribebulk**);**

addReply**(**c**,**shared**.**nullbulk**);**

addReplyLongLong**(**c**,**dictSize**(**c**->**pubsub\_channels**)+**

listLength**(**c**->**pubsub\_patterns**));**

**}**

dictReleaseIterator**(**di**);**

**return** count**;**

**}**

轮训字典c->pubsub\_channels，针对其中的每一个channel，调用函数pubsubUnsubscribeChannel，使客户端退订该频道，并将退订的频道数记录到count中；

最后，如果notify非0，并且count为0，则回复客户端相应信息；

pubsubUnsubscribeChannel函数用于客户端退订单个频道，该函数的代码如下：

int pubsubUnsubscribeChannel**(**redisClient **\***c**,** robj **\***channel**,** int notify**)** **{**

dictEntry **\***de**;**

list **\***clients**;**

listNode **\***ln**;**

int retval **=** 0**;**

*/\* Remove the channel from the client -> channels hash table \*/*

incrRefCount**(**channel**);** */\* channel may be just a pointer to the same object*

*we have in the hash tables. Protect it... \*/*

**if** **(**dictDelete**(**c**->**pubsub\_channels**,**channel**)** **==** DICT\_OK**)** **{**

retval **=** 1**;**

*/\* Remove the client from the channel -> clients list hash table \*/*

de **=** dictFind**(**server**.**pubsub\_channels**,**channel**);**

redisAssertWithInfo**(**c**,NULL,**de **!=** **NULL);**

clients **=** dictGetVal**(**de**);**

ln **=** listSearchKey**(**clients**,**c**);**

redisAssertWithInfo**(**c**,NULL,**ln **!=** **NULL);**

listDelNode**(**clients**,**ln**);**

**if** **(**listLength**(**clients**)** **==** 0**)** **{**

*/\* Free the list and associated hash entry at all if this was*

*\* the latest client, so that it will be possible to abuse*

*\* Redis PUBSUB creating millions of channels. \*/*

dictDelete**(**server**.**pubsub\_channels**,**channel**);**

**}**

**}**

*/\* Notify the client \*/*

**if** **(**notify**)** **{**

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**unsubscribebulk**);**

addReplyBulk**(**c**,**channel**);**

addReplyLongLong**(**c**,**dictSize**(**c**->**pubsub\_channels**)+**

listLength**(**c**->**pubsub\_patterns**));**

**}**

decrRefCount**(**channel**);** */\* it is finally safe to release it \*/*

**return** retval**;**

**}**

首先从字典c->pubsub\_channels中，删除以channel为key的字典项；

然后在字典server.pubsub\_channels中，寻找以channel为key的字典项de，并从de中得到订阅该channel的客户端列表clients，然后在列表clients中找到客户端c对应的元素ln，将ln从clients中删除；如果clients长度为0，则从字典server.pubsub\_channels中删除该channel；

最后，如果参数notify非0，则回复客户端相应信息；

2：PUNSUBSCRIBE命令

客户端发送PUNSUBSCRIBE命令，用于退订之前通过PSUBSCRIBE命令订阅的频道模式。PUNSUBSCRIBE命令的处理函数是punsubscribeCommand，代码如下：

void punsubscribeCommand**(**redisClient **\***c**)** **{**

**if** **(**c**->**argc **==** 1**)** **{**

pubsubUnsubscribeAllPatterns**(**c**,**1**);**

**}** **else** **{**

int j**;**

**for** **(**j **=** 1**;** j **<** c**->**argc**;** j**++)**

pubsubUnsubscribePattern**(**c**,**c**->**argv**[**j**],**1**);**

**}**

**if** **(**clientSubscriptionsCount**(**c**)** **==** 0**)** c**->**flags **&=** **~**REDIS\_PUBSUB**;**

**}**

如果该命令没有任何参数，则调用pubsubUnsubscribeAllPatterns函数，设置该客户端c退订所有频道模式；

否则，针对命令参数中的每一个pattern，调用pubsubUnsubscribePattern，使客户端退订相应频道模式；

最后，如果该客户端c订阅的频道和频道模式数为0，则从客户端标志位中删除REDIS\_PUBSUB标记，表示客户端退出订阅模式；

pubsubUnsubscribeAllPatterns函数用于客户端退订所有频道模式，该函数的代码如下：

int pubsubUnsubscribeAllPatterns**(**redisClient **\***c**,** int notify**)** **{**

listNode **\***ln**;**

listIter li**;**

int count **=** 0**;**

listRewind**(**c**->**pubsub\_patterns**,&**li**);**

**while** **((**ln **=** listNext**(&**li**))** **!=** **NULL)** **{**

robj **\***pattern **=** ln**->**value**;**

count **+=** pubsubUnsubscribePattern**(**c**,**pattern**,**notify**);**

**}**

**if** **(**notify **&&** count **==** 0**)** **{**

*/\* We were subscribed to nothing? Still reply to the client. \*/*

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**punsubscribebulk**);**

addReply**(**c**,**shared**.**nullbulk**);**

addReplyLongLong**(**c**,**dictSize**(**c**->**pubsub\_channels**)+**

listLength**(**c**->**pubsub\_patterns**));**

**}**

**return** count**;**

**}**

轮训列表c->pubsub\_patterns，针对其中的每一个pattern，调用函数pubsubUnsubscribePattern使客户端退订该频道模式，并将退订的频道模式数记录到count中；

最后，如果notify非0，并且count为0，则回复客户端相应信息；

pubsubUnsubscribePattern函数用于客户端退订单个频道模式，该函数的代码如下：

int pubsubUnsubscribePattern**(**redisClient **\***c**,** robj **\***pattern**,** int notify**)** **{**

listNode **\***ln**;**

pubsubPattern pat**;**

int retval **=** 0**;**

incrRefCount**(**pattern**);** */\* Protect the object. May be the same we remove \*/*

**if** **((**ln **=** listSearchKey**(**c**->**pubsub\_patterns**,**pattern**))** **!=** **NULL)** **{**

retval **=** 1**;**

listDelNode**(**c**->**pubsub\_patterns**,**ln**);**

pat**.**client **=** c**;**

pat**.**pattern **=** pattern**;**

ln **=** listSearchKey**(**server**.**pubsub\_patterns**,&**pat**);**

listDelNode**(**server**.**pubsub\_patterns**,**ln**);**

**}**

*/\* Notify the client \*/*

**if** **(**notify**)** **{**

addReply**(**c**,**shared**.**mbulkhdr**[**3**]);**

addReply**(**c**,**shared**.**punsubscribebulk**);**

addReplyBulk**(**c**,**pattern**);**

addReplyLongLong**(**c**,**dictSize**(**c**->**pubsub\_channels**)+**

listLength**(**c**->**pubsub\_patterns**));**

**}**

decrRefCount**(**pattern**);**

**return** retval**;**

**}**

首先，根据pattern，在列表c->pubsub\_patterns中寻找相应的节点ln；然后将ln从列表c->pubsub\_patterns中删除；

然后，根据c和pattern，在列表server.pubsub\_patterns中寻找相应的节点ln，然后将ln从列表server.pubsub\_patterns中删除；

最后，如果参数notify非0，回复客户端相应信息；