**[后端通讯模块]**

**软件详细设计说明书**

目录

[1. 目的 2](#_Toc500945621)

[2. 代码框架描述 2](#_Toc500945622)

[3. API接口使用说明 3](#_Toc500945623)

[3.1 连接与断开连接 3](#_Toc500945624)

[3.2 Redis基本操作 3](#_Toc500945625)

[3.3 订阅与分发 6](#_Toc500945626)

[3.4 业务处理 7](#_Toc500945627)

[4. API接口实现说明 9](#_Toc500945628)

[5.1 连接与断开连接 9](#_Toc500945629)

[5.2 Redis基本操作 10](#_Toc500945630)

[5.3 订阅与发布 11](#_Toc500945631)

[5.4 业务处理 11](#_Toc500945632)

[5.5 clientID说明 12](#_Toc500945633)

[6. 测试用例 13](#_Toc500945634)

[6.1 正常操作 13](#_Toc500945635)

[6.2 多线程操作 13](#_Toc500945636)

[6.3 异常操作 13](#_Toc500945637)

[7. 编译说明 13](#_Toc500945638)

[7.1 Windows下的编译 13](#_Toc500945639)

[7.2 Ubuntu下的编译 14](#_Toc500945640)

# 1. 目的

《后端通讯模块软件详细设计说明书》是ECG后台服务框架总体设计中的后端通讯模块的详细设计说明文档,详细说明了此模块与redis服务的交互、与数据库接入模块的交互以及此模块的详细内部实现，为研发人员设计提供指导性的设计输入文档。

# 2. 代码框架描述



图2-1 Redis服务模块位置

根据ECG后台服务框架的模块划分，后端通讯模块主要位于图2-1中所示的位置上，与Redis之间采用TCP协议通讯，与业务模块之间采用Redis 的订阅功能进行交互。后端通讯模块作为一个链接库供各个模块调用。除了get、set、push、pop等基本操作采用同步通讯的之外，订阅和拉取功能采用了异步通讯的方式，目的是为了将业务处理模块与前端显示进行解耦，降低各个模块之间的耦合性。

后端通讯模块主要封装了Redis服务的get、set、lpush、rpop以及subscribe功能，使用Redis的ae事件库和libevent事件库实现异步通讯，并基于此实现了消息广播以及任务分发功能。内部实现针对订阅的客户端以及任务处理模块建立了一个会话管理。对于只使用基本功能的客户端，此模块只是管理与Redis服务的连接，并代理客户完成相应操作。对于复杂的业务处理，例如订阅和任务分发。此模块实现了数据流传输会话管理以及心跳信令交互会话管理。这种模式可以使得客户端与业务处理模块解耦。当客户端请求量很大的时候，同一批数据可以有多个业务处理模块同时进行处理，只需要注册相同的请求队列、心跳通道以及相应的回调函数即可。可以提高整体的吞吐量。

# 3. API接口使用说明

## 3.1 连接与断开连接

1. bool connect(const std::string & strIp, int iPort, bool bNeedSubs = false);

功能说明：连接redis服务

参数说明：

* const std::string & strIp：redis服务的ip地址
* int iPort：redis服务的侦听端口
* bool bNeedSubs：是否启用redis的键空间通知功能，默认不启用。如果不启用此功能，则sub、pull、侦听客户端get操作等接口将不起作用。

返回值说明：连接成功返回true，失败返回false

备注：如果仅仅需要使用redis的基本功能，建议不启用redis的键空间通知功能。

1. void disconnect();

功能说明：断开连接redis服务

参数说明：无参数

返回值说明：无返回值

备注：此函数的功能是将连接标志置为false，并清空客户端的订阅的消息。

1. 例子

|  |
| --- |
| CRedis\_Utils redis("clientA");  redis.connect("192.168.31.217", 6379, true);  redis.disconnect(); |

## 3.2 Redis基本操作

1. int get(const std::string & strInKey, std::string & strOutResult);

功能说明：执行redis的get key命令

参数说明：

* const std::string & strInKey：redis键，值的类型为字符串
* std::string & strOutResult：返回结果字符串，get结果或者出错信息

返回值说明：操作成功返回0。失败返回值大于0（状态码）。失败原因通过strOutResult查看。如果get操作的key不存在，strOutResult为空字符串。

备注：

1. int set(const std::string & strInKey, const std::string & strInValue, std::string & strOutResult);

功能说明：执行redis的set key value命令

参数说明：

* const std::string & strInKey：redis键，值的类型为字符串
* const std::string & strInValue：键对应的值
* std::string & strOutResult：set成功时一般返回OK，失败时返回出错信息

返回值说明：操作成功返回0。失败返回值大于0（状态码）。

备注：

1. int push(const std::string & strInKey, const std::string & strInValue, std::string & strOutResult);

功能说明：执行redis的lpush key value命令

参数说明：

* const std::string & strInKey：redis键，值的类型为list
* const std::string & strInValue：键对应的值
* std::string & strOutResult：push成功一般返回list的长度，失败时返回出错信息

返回值说明：操作成功返回0。失败返回值大于0（状态码）。

备注：

1. int pop (const std::string & strInKey, std::string & strOutResult);

功能说明：执行redis的rpop key命令

参数说明：

* const std::string & strInKey：redis键，值的类型为list
* std::string & strOutResult：返回结果字符串，pop结果或者出错信息

返回值说明：操作成功返回0。失败返回值大于0（状态码）。失败原因通过strOutResult查看。如果pop操作的key不存在，strOutResult为空字符串。

备注：

1. 例子

|  |
| --- |
| CRedis\_Utils redis("clientA");  redis.connect("192.168.31.217", 6379); //不需要订阅功能  std::string msg;  //set例子  for (int i = 0; i < 10; ++i)  {  std::string key\_ = "hello" + int2str(i);  std::string value = "world" +int2str(i);  if (redis.set(key\_, value, msg) == 0)  DEBUGLOG("set op succ!!! msg = " << msg);  else  ERRORLOG("set op fail!!! err = " << msg);  }  //get例子  for (int i = 0; i < 10; ++i)  {  std::string key\_ = "hello" + int2str(i);  int status = redis.get(key\_, msg);  if(msg.length() == 0)  DEBUGLOG("no data!!! key = " << key\_.c\_str());  else if (status == 0)  DEBUGLOG("get op succ!!! msg = " << msg);  else  ERRORLOG("get op fail!!! err = " << msg);  } |

## 3.3 订阅与分发

1. int subs(const std::string & strInKey, subsCallback cb);

功能说明：订阅键值变化通知，键对应的值类型为字符串。当key对应的值被更新或删除时，所有订阅的客户端都会收到此消息。

参数说明：

* const std::string & strInKey：订阅的键，支持“\*?”等通配符。
* subsCallback cb：有订阅消息时的回调处理函数，函数原型为typedef void(\*subsCallback)(const std::string & strKey, const std::string & strValue);

返回值说明： 订阅成功返回0。失败返回值大于0（状态码）

备注： subsCallback中的key为订阅的键，value为key对应的值。当key被更新时，value为新的值；当key被删除时，value为空字符串。

1. bool unsubs(const std::string & strInKey);

功能说明：取消订阅某个或某类键。

参数说明：

* const std::string & strInKey：取消订阅的键。

返回值说明： 取消订阅成功返回true，失败返回false

备注：只有通过subs接口订阅过的键才会生效，不支持通配符。

1. int pull(const std::string & strInKey, pullCallback cb);

功能说明：订阅键值变化通知，键对应的值类型为list，当key对应的值被更新或删除时，订阅的所有客户端都会收到此消息。

参数说明：

* const std::string & strInKey：订阅的键，支持“\*?”等通配符。
* subsCallback cb：有订阅消息时的回调处理函数，函数原型为typedef void(\*pullCallback)(const std::string & strKey, const std::string & strValue);

返回值说明： 订阅成功返回0。失败返回值大于0（状态码）

备注： pullCallback中的key为订阅的键，value为key对应的值。当key有lpush，value为rpop key的值；当key被删除时，value为空字符串。

1. bool unpull(const std::string & strInKey);

功能说明：取消订阅某个或某类键。

参数说明：

* const std::string & strInKey：取消订阅的键。

返回值说明： 取消订阅成功返回true，失败返回false

备注：只有通过pull订阅过的键才会生效，不支持通配符。

1. 例子

|  |
| --- |
| //订阅回调函数  void subCB(const std::string & strKey, const std::string & strValue)  {  if (value.length() == 0)  DEBUGLOG("key = " << strKey << ", was deleted...");  else  DEBUGLOG("got subcb msg, strKey = " << key << ", value = " << strValue);  }  //订阅例子  CRedis\_Utils redis("clientA");  redis.connect("192.168.31.217", 6379, true); //需要订阅功能  redis.subs("hello\*", subCB); //当hello\*被更新或添加的时候，回调会被执行  std::this\_thread::sleep\_for(std::chrono::milliseconds(10000));  redis.unsubs("hello\*"); //取消订阅hello\*  getchar(); |

## 3.4 业务处理

1. int subsClientGetOp(const std::string & strInKey, clientOpCallBack cb);

功能说明： 注册客户端get操作的key以及回调函数。

参数说明：

* const std::string & strInKey：订阅的key
* clientOpCallBack cb：key被某个客户端get时的回调处理函数，函数原型为typedef void(\*clientOpCallBack)(const std::string & strKey, const std::string & strValue);

返回值说明：订阅成功返回0。失败返回值大于0（状态码）

备注： pullCallback中的key为订阅的键，value为key对应的值。当key有lpush，value为rpop key的值；当key被删除时，value为空字符串。

1. bool unsubClientGetOp(const std::string & strInKey);

功能说明：注销客户端get操作的key，以及对应的请求队列名称，心跳信令名称，处理回调函数。

参数说明：

* const std::string & strInKey：取消订阅的键。

返回值说明： 取消订阅成功返回true，失败返回false。

备注：只有通过subsClientGetOp注册过的键才会生效，不支持通配符。

1. void stopSubClientGetOp();

功能说明：停止监听客户端的get操作

参数说明：无参数

返回值说明： 无返回值

备注：

1. int notifyRlt(const std::string & strInKey, const std::string & strInValue);

功能说明：通知客户端业务处理完成

参数说明：

const std::string & strInKey：业务模块处理的key

const std::string & strInValue：业务模块的处理结果

返回值说明：操作成功返回0。失败返回值大于0（状态码）

备注：业务模块处理完成之后，务必使用此接口通知客户端处理完成。如果使用set接口通知，则可能会导致无法收到之后相同的业务请求。

1. 例子

|  |
| --- |
| //业务处理回调函数  void getCB(const std::string & strKey, const std::string & strValue)  {  DEBUGLOG("clientA got get msg!!!");  CRedis\_Utils redis("A");  redis.connect("192.168.31.217", 6379);  std::string msg;  if (value.length() == 0)  redis.notifyRlt(strKey, "nil", msg);//通知客户端处理完成  else  {  std::string value\_ = strValue + std::string("getCB");  redis.notifyRlt(strKey, value\_, msg); //通知客户端处理完成  }  }  std:string msg;  CRedis\_Utils redis("clientA");  redis.connect("192.168.31.217", 6379, true); //需要订阅功能  redisA.subsClientGetOp("hello\*", getCBA); //注册监听hello\*关键字，当hello\*被名称为clientA的客户端get时，业务处理回调函数会被执行  getchar(); |

# 4. API接口实现说明

## 连接与断开连接

1. 连接

类中保存两个redis上下文，一个是同步的，一个是异步的。当启用redis的键空间通知功能时，同时使用这两个上下文连接redis服务，同步上下文负责redis的基本操作接口；异步上下文负责redis键空间通知的分发。当不启用redis的键空间通知功能时，仅仅使用同步上下文来进行redis的基本操作。

此外还有一个已连接标志位，当成功连接上redis服务的时候，此标志为置为true。重复调用connect接口时，会检查此标志，如果为true，不进行重复连接操作，直接返回。使用其他接口时也会检查此标志，只有连接上redis服务之后才能执行相应的操作。

但是连接redis服务之后，如果长时间不操作的话，redis服务会主动断开连接。当检查到redis主动断开连接之后，会自动重新连接redis服务。连接操作的流程图如下图所示：



图4-1 连接操作流程图

1. 断开连接

断开连接并不是真正的断开与redis服务的连接，只是改变一些标志位和垃圾清理的操作。

## Redis基本操作

概述：初始化的时候，每个客户端都会有一个clientID，客户端在使用本模块的接口操作redis的时候，每个key都会加上clientID前缀，返回的数据将此前缀截断，对于客户端而言，clientID是透明的。加上clientID前缀的目的是用于数据隔离，每个客户端只处理属于自己的数据，各模块之间相互不影响，将可能发生的错误控制在小范围之内。

1. get

客户端执行get key操作，可能redis中并没有此数据，或者此数据已经过时，需要业务模块进行相应的处理之后，才将其返回给客户端。所以在执行get操作之前需要额外的动作，第一步先检查是否业务模块是否可用以及key是否有对应的业务处理模块，如果没有则直接返回结果；如果有则将key塞入请求队列中等待，等待业务模块处理结果。如果超时未处理完成则直接返回旧结果。

实现步骤：

* 检测是否有可用业务处理模块：业务处理模块订阅了某个key之后，会定时更新心跳信令，每一个订阅的key都会有相应的心跳信令。检测是否有可用的业务处理模块，只需要检测对应的心跳信令即可是否存在即可。
* 将请求塞入请求槽中
* 超时处理

一些实现技巧：

* 关于订阅key的引用计数
* 关于哨兵

1. set

检查标志位，然后将key加上clientID前缀，执行redis的set key value指令。

1. push

检查标志位，然后将key加上clientID前缀，执行redis的lpush key value指令。

1. pop

检查标志位，然后将key加上clientID前缀，执行redis的rpop key value指令。

## 订阅与发布

1. subs

客户端注册要订阅reids的键以及相应的回调函数，键对应的值类型是字符串。服务模块使用键值对（key🡪callback）来管理已订阅的键，并利用redis的键空间通知功能获得被修改或删除的键；当该键发生set操作，将此键和新的值返回给客户端；当此键发生del操作时，将此键返回给客户端，值为空字符串。

1. pull

客户端注册要订阅reids的键以及相应的回调函数，键对应的值类型是list。服务模块使用键值对（key🡪callback）来管理已订阅的键，并利用redis的键空间通知功能获得被插入或删除的键；利用redis的键空间通知功能，当该键发生插入操作，将此键和pop出来的值返回给客户端；当此键发生del操作时，将此键返回给客户端，值为空字符串。

1. unsub

注销客户端已订阅的键及回调函数，即将键值对中的相应信息删除。当该键发生set或del操作时，不再通知客户端。

1. unpull

注销客户端已订阅的键及回调函数，即将键值对中的相应信息删除。当该键发生插入或del操作时，不再通知客户端。

## 业务处理

1. subsClientGetOp

业务处理模块专用函数，侦听客户端的get key操作。和订阅的函数一样，此函数总共有2个参数，详见3.4。成功注册监听之后，此模块会维护一些列请求队列和心跳信令。用户不需要关心心跳信令的管理，成功注册之后，此模块会自动定时更新心跳信令，注销监听此key之后，将不会更新对应的心跳信令，原则上每一个监听的key对应一个心跳信令。请求队列用于存放客户端get操作的key，然后进行任务分发。整体流程：业务模块注册key和回调函数。当客户端发生get操作时，如果key已有业务模块订阅处理，此模块将此key塞入至相应的请求队列，利用redis的键空间通知功能将任务分发至空闲的业务模块进行处理。

如果同一类数据请求量很大时，可以同时有多个业务模块进行处理。只需要将此函数的各个参数一致即可，当客户端产生get操作时，该key会被塞入此请求队列中。通过此模块的发布功能，将任务分发至空闲的处理模块进行数据处理即可。

1. unsubClientGetOp

注销业务模块已订阅的key和回调函数。当客户端get该key时不通知该业务模块进行处理。

1. stopClientGetOp

注销业务模块所有已订阅的key和回调函数。当客户端发生get操作时不通知该业务模块进行处理。

## clientID说明

此模块未提供默认构造函数，初始化时必须传入clientID的值，如果传入空字符串，默认

clientID为default。此模块的所有接口都会对传入的key都会加上clientID前缀，返回的key则会去掉client前缀。

加上clientID之后，A客户端订阅keyA，B客户端订阅keyA，对于两个客户端而言看起来是订阅了同一个键keyA；但是实际上A客户端订阅的时AkeyA，B客户端订阅的是BKeyA，这样就实现了数据隔离。同理get、set等其他操作也是一样的。

# 5. 测试用例

## 正常操作

## 多线程操作

## 异常操作

1. 空指针
2. 空字符
3. 特殊字符：注意转义字符以及%%
4. ip或端口错误
5. 网络异常中断

# 6. 编译说明

## Windows下的编译

编译环境：win10 Visual Studio2015

依赖库：hiredis log4cplus

编译步骤：

1. Hiredis的编译

* 下载：<https://github.com/MicrosoftArchive/redis> 目前使用的版本为3.0
* 编译：解压之后打开msvc/RedisServer.sln，直接生成hiredis和Win32\_Interop两个模块，在msvs/x64/Debug/下得到Win32\_Interop.lib和hiredis.lib
* 头文件：deps/hiredis的所有文件，src/的所有文件

1. Log4cplus的编译

* 下载地址：<https://sourceforge.net/projects/log4cplus/files/log4cplus-stable/1.2.0/> 目前使用的版本为1.2.0
* 编译：解压之后打开msvc10/ log4cplus.sln，选择平台x64，修改log4cplus模块的属性，将配置属性-常规的字符集更改为使用Unicode字符集，然后生成log4cplus模块，在msvc10/x64/bin.Debug下得到log4cplusD.lib和log4cplusD.dll
* 头文件：include下的所有文件

1. 后端通讯模块的编译
   * 依赖库准备：在C盘根目录新建hiredis/include和hiredis/lib文件夹，include存放上述的hiredis头文件，lib存放上述的hiredis的链接库。同理log4cplus一样。修改工程的属性，包括配置属性-c/c++-常规-附加包含目录，包含hiredis和log4cplus的头文件位置，例如C:\log4cplus\include、C:\hiredis\deps、C:\hiredis\src；还有配置属性-库管理-常规-附加库目录和附加依赖项，包含静态链接库的位置和名称。例如附加库目录：C:\log4cplus\lib、C:\hiredis\lib，附加依赖项：hiredis.lib、Win32\_Interop.lib、log4cplusD.lib。
   * 编译：直接生成libXglRedis模块，得到libXglRedis.lib静态链接库。
   * 头文件include下所有文件以及common\_tool.h、Redis\_Utils.h和RedisRPC.h
   * 使用：添加上述头文件，按照上述依赖库准备，在添加上libXglRedis.lib，使用的时候只需包含Redis\_Utils.h头文件即可。使用方法详见*API接口使用说明*。

## Ubuntu下的编译

编译环境：Ubuntu 16.04 GNU ar (GNU Binutils for Ubuntu) 2.26.1 gcc version 5.4.0

依赖库：hiredis log4cplus libevent

1. Hiredis编译

* 下载：<http://download.redis.io/releases/redis-4.0.2.tar.gz>
* 编译：解压，进入hiredis目录，执行make&&make test&&make install&&ldconfig

1. Log4cplus编译
   * 下载：<https://sourceforge.net/projects/log4cplus/files/log4cplus-stable/1.2.0/>
   * 编译：解压，./configure CXXFLAGS="-std=c++0x"&&make&&make install&&ldconfig
2. Libevent编译
   * 下载：[https://github.com/libevent/libevent/releases/download/release-2.1.8-stable/libevent- 2.1.8-stable.tar.gz](https://github.com/libevent/libevent/releases/download/release-2.1.8-stable/libevent-%202.1.8-stable.tar.gz)
   * 编译：解压，configure --prefix=/usr&&make&&make install
3. 后端通讯模块的编译

进入libXglRedis/目录，make&&make install