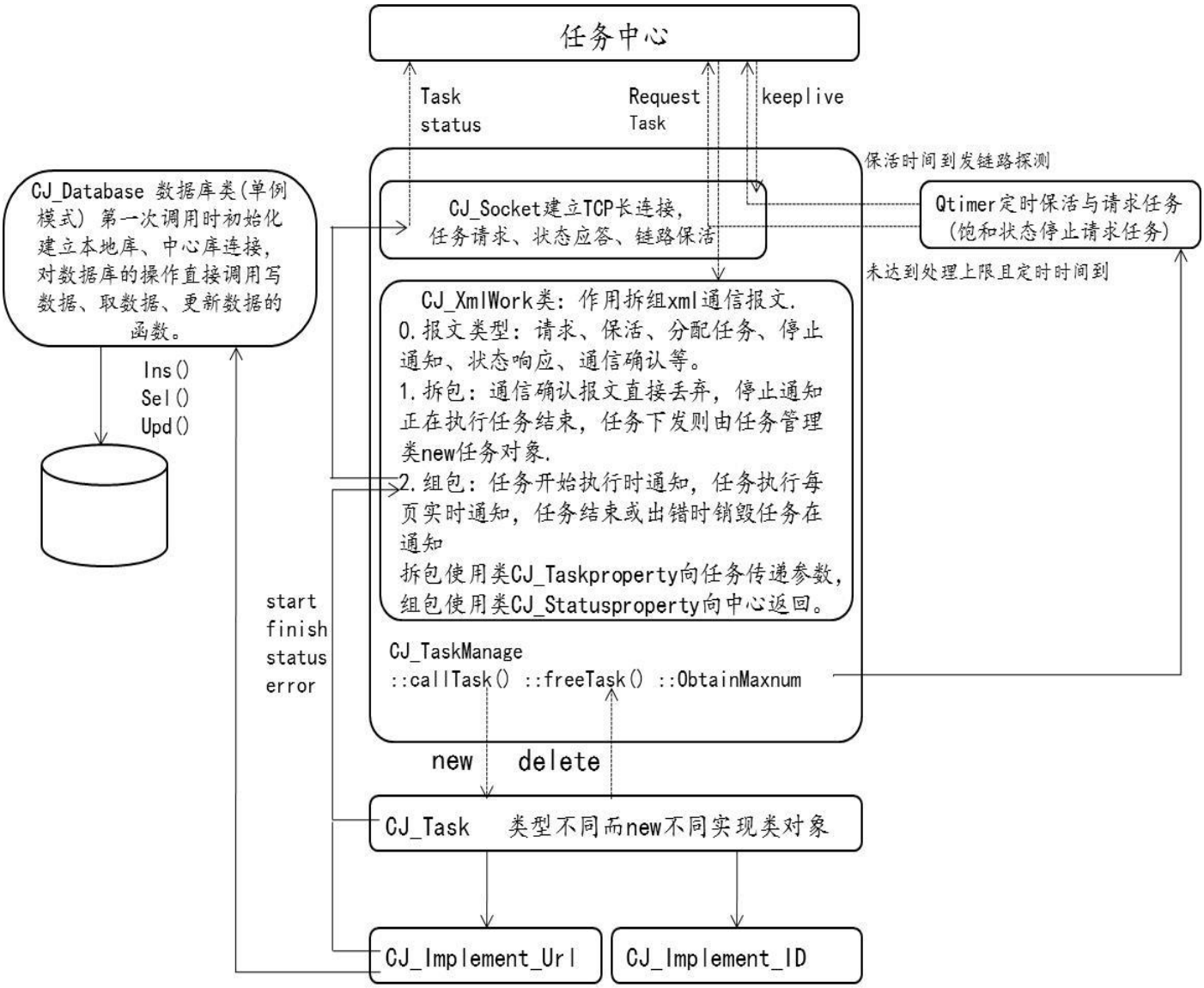
目录

1. 采集客户端说明

1.1 采集功能介绍

采集客户端是获取其他同类网站有价值数据的一个工具，主要通过搜索关键字，友情链接等方法查找网站，请求 http 链接并获的网页源码数据，按获取要求控制并处理数据最终获得完整的房产信息源，此客户端为公司网站【城市房产】提供基础数据，主要负责搜索信息，包括楼盘、二手房、租房等；维护各个城市数据稳定；完善城市数据等。

1.2 采集客户端流程图



2. 采集客户端类定义

2.1 类的说明

类名	描述
CJ_Database	数据库操作类：单例模式创建数据库连接，由 IP 和库名唯一决定链接，数据操作时先获取数据库连接（不存在即创建）再对数据库增、删、改、查等操作。
CJ_Dataget_Origin	房源信息类： 主要是获得房源信息，将信息要素存放在此类对象中，一个对象保存一个房源信息记录，使用获得到的对象数据方便 查询、插入数据库等。
CJ_Fieldproperty	任务模板类： 主要是从任务分配中心获取，模板字段详细信息存放在此类对象中，此类也是数据存放容器类，一个任务下发会包含多个此类对象说明每个要获取要素控制说明信息。
CJ_Implementtask	任务基类：定义多种类型任务，处理流程中使用的虚函数，此类并未具体实现方法。
CJ_Implementtask_ID	ID 自增型任务类（任务类型一）：使用有递增变化规律的 url 地址，获取网页源码从中提取有价值信息。
CJ_Implementtask_URL	URL 列表任务类（任务类型二）： 使用 url 地址获取各大网站展示列表，并设置翻页，获取各个展示列表中的 url 并存放到索引信息数据库。
CJ_Implementtask_URL_detail	URL 详细任务类（任务类型三）： 使用索引信息库 url 列表任务得到的 url 访问网页获得源码从中提取有价值信息。
CJ_Sericedatageturlinde	URL 索引信息类：主要是存放 URL 索引信息类，将信息要素存放在此类对象中，一个对象保存一个索引信息记录，可以方便的使用对象与数据库交互。
CJ_Socket	通信类： 此类使用 socket tcp 长链接与任务分配中心通信，包括任务请求、链路保活、任务执行实时响应、链接中断续链、中断超时后处理、及不完整包处理。
CJ_Statusproperty	任务响应类：主要是从任务执行中，实时获取数据并存放在此类对象中，使用对象中变量值组响应报文通知任务分配中心。
CJ_Task	任务类：负责任务开始（根据任务类型判断）、结束、停止资源回收等工作
CJ_TaskManager	任务管理类：整个客户端最重要部分，负责创建其他类对象、管理其它类对象交互、定义采集客户端处理流程走向等工作。
CJ_Taskproperty	任务属性类： 主要是从任务分配中心获取，任务字段详细信息存放在此类对象中，此类也是数据存放容器类，一个任务下发只对应一个此类对象，说明此任务的控制说明信息。
CJ_XmlWork	报文拆组包类：处理 XML 报文解析与组装，处理多包一次到情况。

2.1 任务管理类

CJ_TaskManager:

```
#ifndef CJ_TASKMANAGER_H
#define CJ_TASKMANAGER_H
```

```
#include <QObject>
#include <QTcpSocket>
#include <QDebug>
#include <QtCore>
#include <QtXml>
#include <QTimer>
#include "cj_xmlwork.h"
#include "cj_task.h"
#include "cj_socket.h"
#include "cj_taskproperty.h"
#include "cj_statusproperty.h"

#define     MAXTASKNUM      10          //定时申请间隔
#define     APPLY_INTERVAL  5000        //定时申请间隔
#define     KEEPL_INTERVAL  20000       //定时保活间隔

class CJ_TaskManager : public QObject
{
    Q_OBJECT
public:
    explicit CJ_TaskManager(QObject *parent = 0);
    static   QMap<QString, CJ_Task*>    mapTask;
    CJ_Socket       *socket;            //Socket 对象
    CJ_XmlWork      *xmlwork;           //XML 拆组对象
    CJ_Task         *task;              //任务对象
    QTimer          *timer1;            //定时器一 探测链路
    QTimer          *timer2;            //定时器二 请求任务
signals:
    void  taskResponse(CJ_Statusproperty& );
    void  taskRequest();
public slots:
    void  callTask(CJ_Taskproperty&);    //创建执行任务
    void  freeTask(CJ_Statusproperty&);  //释放销毁任务
    void  stopTask(CJ_Taskproperty&);    //停止单个任务
    void  stopAllTask();                 //停止全部任务
    void  checkCurrentTasknum();         //当前任务数  未达到即申请
};
#endif // CJ_TASKMANAGER_H
```

2.2 通信 Socket 类

CJ_Socket:

```
#ifndef CJ_SOCKET_H
#define CJ_SOCKET_H

#include <QTcpSocket>
```

```
#include <QHostAddress>
#include <QDateTime>
#include <QTimer>

#define  CENTER_IP       "10.10.15.222"  //IP
#define  CENTER_PORT     5450            //端口
#define  INTERVAL_LINK   5000            //重连间隔
#define  MAX_TRY_LINK    4               //重连次数

class CJ_Socket : public QObject
{
    Q_OBJECT
public:
    explicit CJ_Socket(QObject *parent = 0);
    QTcpSocket *getRw_socket() const;
    void        acquireCompletepackage(QByteArray&, QByteArray&);

private:
    int         repeatnum;          //当前重链次数
    QTimer      *timer;
    QTcpSocket  *rw_socket;
    qint64      packbytes;          //报文大小
    qint64      oncebytes;          //一次写入大小
    qint64      remainbytes;        //剩余大小
    QByteArray  readbuff;           //读取缓冲区
    QByteArray  writebuff;          //写入缓冲区
signals:
    void readFinished(QByteArray);  //读到数据 触发信号交给拆包
    void connectCreated();          //套接字已连接 触发信号申请任务.
    void connectBreaked();          //链接中断 触发信号终止所有任务.
public slots:
    void newConnection();
    void readDataFromServer();
    void writeDataToServer(QByteArray&);
    void writeDataAgainToServer(qint64);
    void processError(QAbstractSocket::SocketError );
    void disconnectHandle();
    void connectFinished();
};
#endif // SOCKET_H
```

2.3XML 报文拆组类

CJ_XmlWork:

```
#ifndef CJ_XMLWORK_H
#define CJ_XMLWORK_H

#include <QtCore>
#include <QObject>
#include <QtDebug>
#include <QtXml>
#include <QHostInfo>
#include <QHostAddress>
#include "cj_socket.h"
#include "cj_taskproperty.h"
#include "cj_statusproperty.h"
#include "cj_fieldproperty.h"

#define   LABEL_TASK     "task"
#define   LABEL_PATTERN  "taskpattern"
#define   LABEL_FIELD    "field"

class CJ_XmlWork : public QObject
{
    Q_OBJECT
public:
    CJ_XmlWork(QObject *parent = 0);
    CJ_XmlWork(CJ_Socket *socket);
    static          long    id;             //唯一标识:reqid_%ld
    CJ_Socket       *socket;                //获得 socket 状态
    void            getIpGetway(QString&, QString&);
    void            multiPackageSplit(QByteArray&, QList<QByteArray>&);
signals:
    void            createPackagefinish(QByteArray& );          //组返回报文完成
    void            parseTaskfinish(CJ_Taskproperty&);
    void            parseStopfinish(CJ_Taskproperty&);
public slots:
    void            createRequestPackage();                     //请求任务报文组包
    void            createKeeplivPackage();                     //链路保活报文组包
    void            createResponePackage(CJ_Statusproperty&);   //任务响应报文组包
    void            parsePackage(QByteArray);                   //拆包

};
#endif // XMLWORK_H
```

2.4 任务类

CJ_Task:

```
#ifndef CJ_TASK_H
```

```
#define CJ_TASK_H

#include <QObject>
#include "cj_taskproperty.h"
#include <QString>
#include <QDebug>
#include "cj_statusproperty.h"
#include "cj_statuspropertyadded.h"

class CJ_Implementtask;
class CJ_Task : public QObject
{
    Q_OBJECT
public:
    explicit CJ_Task(QObject *parent = 0);
    CJ_Task(CJ_Taskproperty taskpro);
    ~CJ_Task();
    CJ_Taskproperty getTaskpro();
    void        updateStatusproperty(CJ_Statusproperty &);//状态响应
    void        startTask();        //开始
    void        stopTask();         //停止
    void        recyTask();         //回收
private:
    CJ_Taskproperty     taskpro;
    CJ_Implementtask    *task_impl;
signals:
    void        status(CJ_Statusproperty&);
};
#endif // TASK_H
```

2.5 数据库操作类

CJ_Database:

```
#ifndef CJ_DATABASE_H
#define CJ_DATABASE_H
#define SQLSERVER       //MYSQL

#include <QObject>
#include <QtCore>
#include <QDebug>
#include <QSqlDatabase>
#include <QSqlError>
#include <QSqlQuery>
#include "cj_dataget_origin.h"
#include "cj_datageturlidx.h"
```

```
#include "cj_taskproperty.h"
#include "cj_sericedatageturlinde.h"

typedef  QPair<QString,QString>  Pair;
class CJ_Database : public QObject
{
    Q_OBJECT
private:
    CJ_Database(int , CJ_Taskproperty&);
    ~CJ_Database();
    QSqlDatabase            db;
    static   CJ_Database*   GetInstance(int , CJ_Taskproperty&);//new db_Instance.
    static   CJ_Database    *db_Instance;
    static   QMap<QString, CJ_Database*>  map;//存不同链接的 db_Instance.
                                              由 Qstring 唯一区分【IP+dbname】
public:

    static   double   DbMathfuncOnUrlindex(CJ_Taskproperty&,  QString&);
    static   QList<CJ_Dataget_Origin>       DbSelectOnInformation(CJ_Taskproperty&,
QString&);
    static   QList<CJ_Sericedatageturlinde> DbSelectOnUrlindex   (CJ_Taskproperty&,
QString&);
    static   int      DbInsertOnInformation(CJ_Taskproperty&,
QList<CJ_Dataget_Origin>&);
    static   bool     DbInsertOnInformation(CJ_Taskproperty&, CJ_Dataget_Origin&);
    static   int      DbInsertOnUrlindex    (CJ_Taskproperty&,
QList<CJ_Sericedatageturlinde>&);
    static   bool     DbInsertOnUrlindex    (CJ_Taskproperty&,
CJ_Sericedatageturlinde&);
    static   bool     DbUpdateOnInformation(CJ_Taskproperty&, CJ_Dataget_Origin&,
QString&, QString&);
    static   bool     DbUpdateOnUrlindex    (CJ_Taskproperty&, QString &sql);
    static   bool     DbDeleteOnInformation(CJ_Taskproperty&, QString&);
    static   bool     DbDeleteOnUrlindex    (CJ_Taskproperty&, QString&);
};
#endif // DATABASE_H
```

2.6 数据存放<模板 T>类

具有类似相同格式的 存放数据类：
CJ_Taskproperty: CJ_Dataget_Origin: CJ_Fieldproperty: CJ_Sericedatageturlinde: CJ_Statusproperty:

```
#ifndef CJ_TDATA_H
#define CJ_TDATA_H

#include <QObject>
```

```
#include <QtCore>

class CJ_Tdata : public QObject
{
    Q_OBJECT
    Q_PROPERTY(int          var1)
    Q_PROPERTY(QString      var2)
    .........
public:
    CJ_Tdata &operator=(const CJ_Tdata &);
    CJ_Tdata (const CJ_Tdata &);
    explicit CJ_Tdata ();
    void        debug();        //日志函数
    void        clear();        //清除函数
    void        get_typnm(QMap<QString,QString> &);          //获取变量名称与类型
    void        get_typnm(QList<QPair<QString, QString> > &);

    int         var1;
    QString     var2;
    .........
    Q_INVOKABLE int getVar1() const;                //取值
    Q_INVOKABLE void setVar1(int value);            //赋值
    Q_INVOKABLE QString getVar2() const;
    Q_INVOKABLE void setVar2(const QString &value);
    .........
};
#endif // CJ_TDATA_H
```

2.7 任务实现基类

CJ_Implementtask:

```
#ifndef CJ_IMPLEMENTTASK_H
#define CJ_IMPLEMENTTASK_H

#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkReply>
#include <QNetworkRequest>
#include "cj_database.h"
#include "cj_taskproperty.h"
#include "cj_statusproperty.h"
#include "cj_task.h"
#include "cj_fieldproperty.h"

class CJ_Implementtask : public QObject
```

```
{
    Q_OBJECT
public:
    explicit CJ_Implementtask(QObject *parent = 0);
    virtual  ~CJ_Implementtask()=0;
    virtual int     processData(QString &,CJ_Fieldproperty *);//处理数据
    virtual void    startImplement();       //开始任务
    virtual void    stopImplement();        //停止任务

public slots:
    virtual void    receiveData();          //接收数据
    virtual void    sendRequest();          //发送请求
protected:
    QNetworkAccessManager *manager;
    QNetworkRequest *request;
    QNetworkReply   *reply;
    CJ_Task         *task;
    CJ_Taskproperty taskproperty;
};
#endif // IMPLEMENTTASK_H
```

2.8 任务实现派生类

```
#ifndef CJ_IMPLEMENTTASK_*_H
#define CJ_IMPLEMENTTASK_*_H
#include "cj_implementtask.h"
#include "cj_task.h"
#include "cj_statusproperty.h"

class CJ_Implementtask_* : public CJ_Implementtask
{
    Q_OBJECT
public:
    explicit CJ_Implementtask_*(QObject *parent = 0);
    CJ_Implementtask_*(CJ_Task *task);
    ~CJ_Implementtask_*();
    int         processData(QString &);             //处理数据
    void        startImplement();                   //开始任务
    void        stopImplement();                    //停止任务
    void        fieldsSortOnfieldno(QList<CJ_Fieldproperty> &, CJ_Fieldproperty &);
                                                    //模板按编号排序

public slots:
    void        receiveData();                      //接收数据
    void        sendRequest();                      //发送请求
```

```
    void        loginProcess();                     //登陆处理

private:
    CJ_Dataget_Origin           dataget_origin;     //房源存放数据对象
    QList<CJ_Dataget_Origin>    dataget_origins;    //房源存放列表（一次存入行数决定）
    QList<CJ_Fieldproperty>     fieldpropertys;     //模板编号排序
    CJ_Fieldproperty            field;              //(url_list) 列表任务第四个字段
    CJ_Sericedatageturlinde        sericedatageturlinde;      //索引存放数据对象
    QList<CJ_Sericedatageturlinde> sericedatageturlinde_list; //索引存放列表
    CJ_Statusproperty           statuspro;          //状态属性

    int         whichone;                           //（url_detail）当前第几个 url
    QString     beforetime;                         //上次时间
    int         retrycount;                         //重复次数
    int         depth;                              //探测深度
    int         beforeid;                           //前一有效页 id
    int         currentid;                          //当前 id
    int         idstep;                             //步长
    int         totalurl;                           //Url 获取记录总数
    int         storenum;                           //实际存储数  存储数<=采集数
    bool        termbool;                           //请求开关

};
#endif // CJ_IMPLEMENTTASK_*_H
```

3. 采集客户端类关联

3.1 处理流程说明

主程序中 main 函数定义 任务管理 taskmanager 对象，由任务管理 taskmanager 构造函数初始化 通信 socket 对象、XML 协议拆组包对象等、socket 通信建立 tcp 长链接同时接收和发送数据到任务分配中心，当未达到任务最大处理数，则向中心申请任务，得到中心任务响应拆解任务，并通过任务管理开始任务函数 new 新任务，根据任务类型判断具体实现哪种类型的任务，处理任务时实现任务会实时响应数据通知任务管理，等到响应任务管理会实时判断处理最终组包，发送数据到任务分配中心告知任务执行状态。一次处理流程如上: 处理中还涉及到通信连接中断续链、超时停止工作、不完整报处理、多任务包处理、主动任务停止、任务接收回收等操作，在此不做详细描述，详情参见相关代码。

3.2 类之间的关联

信号与槽：

CJ_Taskmanager:

```
connect(socket, SIGNAL(connectCreated()),xmlwork, SLOT(createRequestPackage()));
//通信建立连接信号通知 xmlwork 组链接创建后第一个申请报文

connect(timer1, SIGNAL(timeout()),  xmlwork, SLOT(createKeeplivPackage()));
//定时器 每隔一段时间触发信号通知 xmlwork 组链路保活报文

connect(timer2, SIGNAL(timeout()),  this, SLOT(checkCurrentTasknum()));
//定时器 每隔一段时间触发信号通知 taskmanager 判断最大任务数

connect(this,   SIGNAL(taskRequest()),  xmlwork, SLOT(createRequestPackage()));
//taskmanager  判断若未达到最大任务数则信号通知 xmlwork 组任务申请报文

connect(this,   SIGNAL(taskResponse(CJ_Statusproperty&)),
                xmlwork,SLOT(createResponePackage(CJ_Statusproperty&)));
//taskmanager  接收任务处理响应 触发信号通知 xmlwork 组状态响应报文

connect(socket, SIGNAL(connectBreaked()), this, SLOT(stopAllTask()));
//通信连接中断 并尝试多次重链后仍失败 socket 触发信号通知 taskmanager 停止所有任务

connect(socket, SIGNAL(readFinished(QByteArray)),
                xmlwork,SLOT(parsePackage(QByteArray)));
//socket 接收数据并获得完整包 触发信号通知 xmlwork 可以进行拆包处理

connect(xmlwork,SIGNAL(parseTaskfinish(CJ_Taskproperty&)),
```

```
                                    SLOT(callTask(CJ_Taskproperty&)));
//xmlwork 进行拆包处理 触发信号通知 taskmanager 可以进行任务调用 new task

connect(xmlwork,SIGNAL(parseStopfinish(CJ_Taskproperty&)),
                                    SLOT(stopTask(CJ_Taskproperty&)));
// xmlwork 进行拆包处理 触发信号通知 taskmanager 可以进行任务停止 delete task

connect(xmlwork,SIGNAL(createPackagefinish(QByteArray&)),
                  socket, SLOT(writeDataToServer(QByteArray&)));
// xmlwork 组响应、保活、申请报文完成 触发信号通知 socket 可以进行数据发送到任务分配中心

connect(task,SIGNAL(status(CJ_Statusproperty&)),SLOT(freeTask(CJ_Statusproperty&)));
// 任务状态有变化时 task 触发信号通知 taskmanager 进行信号接收后续处理
```

CJ_Socket:

```
connect(rw_socket,SIGNAL(connected()), this, SLOT(connectFinished()));
// Qtcpsocket 通信连接建立后 触发信号 通知自身 并将定时器关闭、重连次数置 0 等

connect(rw_socket,SIGNAL(connected()), this, SIGNAL(connectCreated()));
// Qtcpsocket 通信连接建立后 触发信号 通知自身 继续传递信号通知 taskmanager

connect(rw_socket,SIGNAL(disconnected()),this,SLOT(disconnectHandle()));
// Qtcpsocket 通信连接中断后 触发信号 通知自身 进行中断续链操作(重链次数之内定时尝试)

connect(rw_socket,SIGNAL(readyRead()),this,SLOT(readDataFromServer()));
// Qtcpsocket 通信数据到达后 触发信号 通知自身 进行数据接收处理操作

connect(rw_socket,SIGNAL(bytesWritten(qint64)),this,SLOT(writeDataAgainToServer(qint64)
));
// Qtcpsocket 通信数据写入后 触发信号 通知自身 看是否还有未发完的数据 未完待发

connect(rw_socket, SIGNAL(error(QAbstractSocket::SocketError)),
                          SLOT(processError(QAbstractSocket::SocketError)));
// Qtcpsocket 通信连接出错后 触发信号 通知自身 进行出错后处理

connect(timer,SIGNAL(timeout()),this,SLOT(disconnectHandle()));
//定时器  每隔一段时间触发信号通知自身 进行多次重复尝试续链(前提已接收到 disconnected 信号 未重新连上)
```

CJ_Implementtask_* :

```
connect(reply,SIGNAL(finished()),this,SLOT(receiveData()));
//http 请求后 有响应到达时 触发信号 进行数据接收处理
```

```
connect(reply,SIGNAL(finished()),this,SLOT(loginProcess()));
//http 请求后 有响应到达时 触发信号 进行数据接收处理
```