CCompare 二次开发接口说明

ccompare.dll 支持对比文件的二次开发功能，对文本格式文件进行对比、同步功能。

环境说明

开发库环境说明：QT (后续也可以推出非 QT 版本，有需求请联系我们 QQ 757210198）
测试库版本的开发环境为：QT5.12.10 ，编译器 VS(2017 v141 社区版本) 。理论上只需要你的开发 QT 环境的版本和 CCompare.dll 的库版本一致就行。VS 版本在测试时是使用的 vs2017 (v141)工具链。

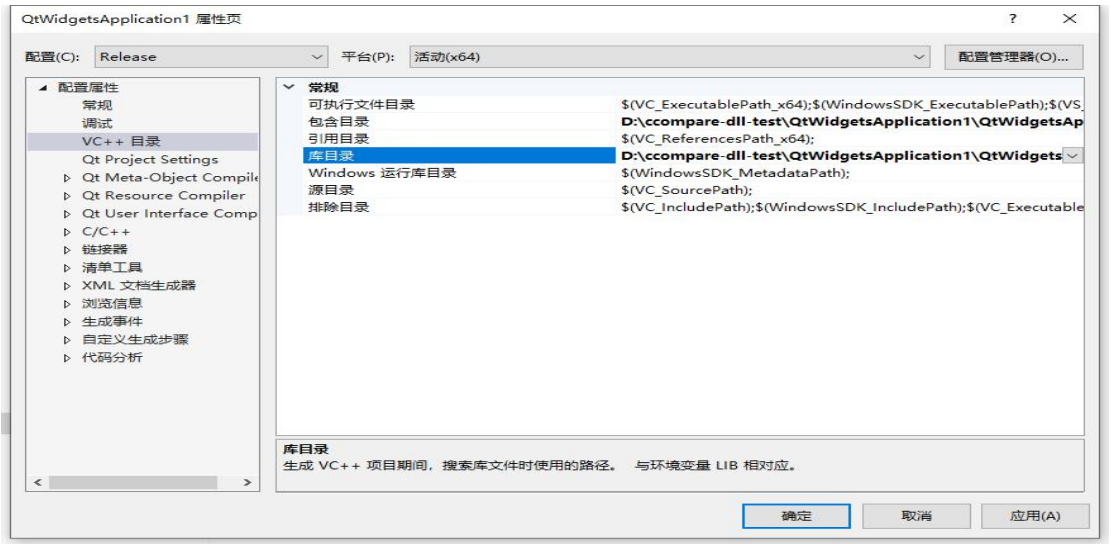
我们会提供和您匹配的 QT 版本的 ccompare.dll 二进制库。如果需要，所有源码均可以付费购买，联系 QQ 757210198。

在测试开发时，要注意您当前使用的 qt 版本，与当前二进制 ccompare.dl 编译的 qt 版本一致；否则编译连接可能有问题。如果您的 qt 版本不是 qt5.12.10，可以先安装 qt5.12.10 测试通过。再找我们购买特定 qt 版本的商业版本 ccompare.dll。

测试代码例子路径：https://gitee.com/cxasm/cc-compare.git

vs 环境配置：

测试工程的 ccompare 目录下，有 ccmp.h blockuserdata.h ccompare.dll ccompare.lib 四个文件，是 release 版本的二次开发库文件和头文件。在 vs 中设置好包含目录和库目录，均设置到例子代码中的 ccompare 目录即可。如下图 vs 配置所示。



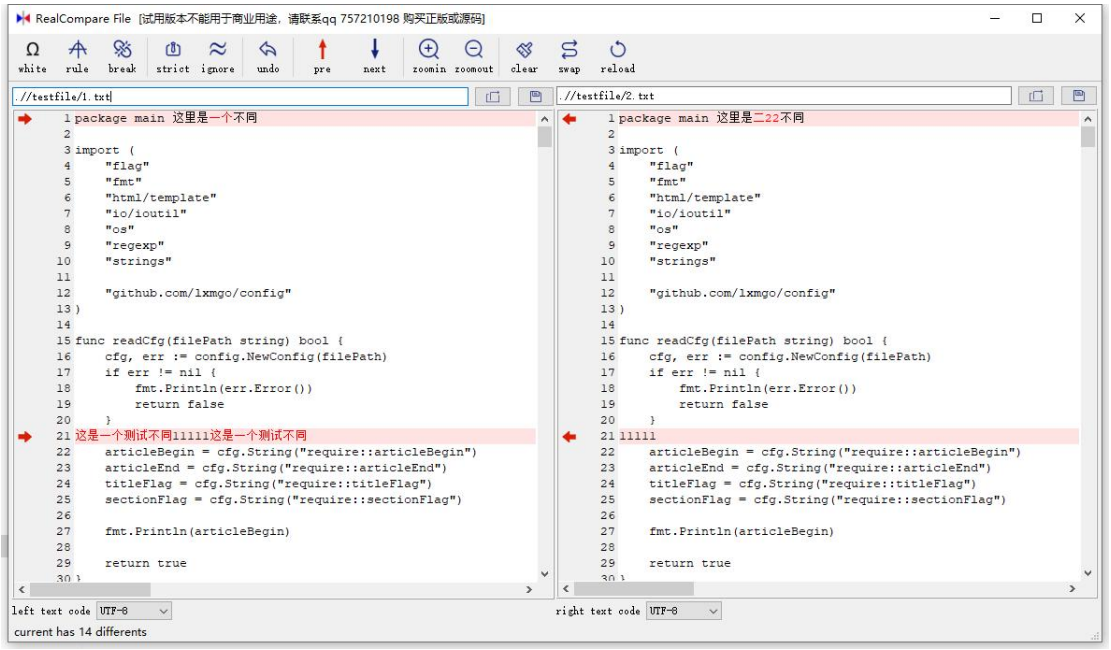
功能介绍

1 对比同步功能

见代码中 compareSyncFile 函数，只需要调用该接口，输入需要处理的两个文件路径参数，则会自动对比完毕并展示。在弹出的窗口中可进行同步操作。这种场景比较傻瓜化，对比和同步界面直接完成，用于在需要对比和同步展示的场景。例子代码中测试运行如下：

```
4
5   QtWidgetsApplication1::QtWidgetsApplication1(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       ui.setupUi(this);
9
10      CCmp* pcmp = new CCmp(this);
11
12  #if 1
13      //文件同步的调用方式，会弹出对比界面，可进行同步
14      pcmp->compareSyncFile(QString(".//testfile/1.txt"), QString(".//testfile/2.txt"));
15  #else
16      //文件对比的方式，自己获取不同点后，进行自定义渲染显示。比较完成后，会发出cmpResult信号
17      connect(pcmp, &CCmp::cmpFinished, this, &QtWidgetsApplication1::on_cmpSuccess);
18      pcmp->compareFile(QString(".//testfile/1.txt"), QString(".//testfile/2.txt"));
19  #endif
20  }
21
```

对比同步运行界面后如下：



如果您对同步界面有定制化需求，请购买源码或者找我们二次开发。

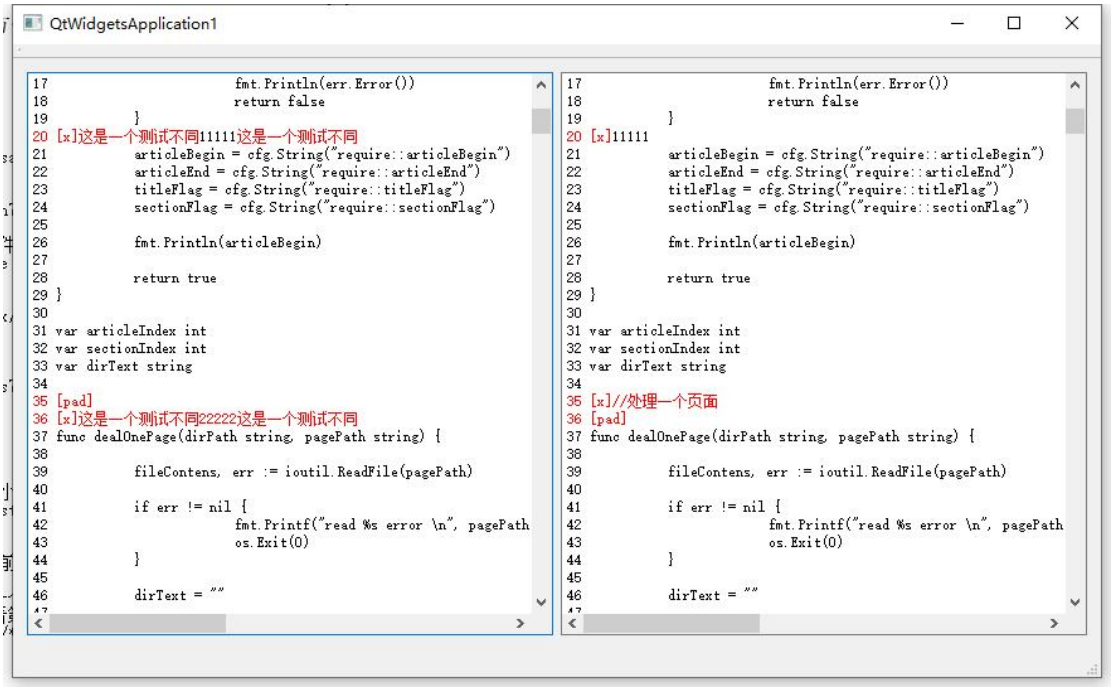
2 对比功能

见代码中 compareFile 函数，只需要调用该接口，输入需要处理的两个文件，则会自动对比，并给出对比结果。**用户在拿到对比结果后，自己去做自定义的渲染和展示。复杂的对比过程和算法部分，ccompare.dll 来完成，灵活的展示部分用户自己定义。**这种方式比较灵活，在二次开发中使用场景更加普遍。

在测试例子中，拿到对比结果后，在 on_cmpSuccess 的回调中，做了一个简单的自定义展示。自定义展示规则为：1）相同的部分用黑色展示。2）不同的部分用红色高亮 3）不同行前面使用[x]表示。4）对齐行前面使用[pad]表示。基于这个简单的例子，用户完全可以自定义规则，满足自身的展示效果。例子代码中测试运行如下：

```
4
5   QtWidgetsApplication1::QtWidgetsApplication1(QWidget *parent)
6       : QMainWindow(parent)
7   {
8       ui.setupUi(this);
9
10      CCmp* pcmp = new CCmp(this);
11
12  #if 0
13      //文件同步的调用方式，会弹出对比界面，可进行同步
14      pcmp->compareSyncFile(QString(".//testfile/1.txt"), QString(".//testfile/2.txt"));
15  #else
16      //文件对比的方式，自己获取不同点后，进行自定义渲染显示。比较完成后，会发出cmpResult信号
17      connect(pcmp, &CCmp::cmpFinished, this, &QtWidgetsApplication1::on_cmpSuccess);
18      pcmp->compareFile(QString(".//testfile/1.txt"), QString(".//testfile/2.txt"));
19  #endif
20  }
21
```

对比后，测试例子做自定义展示效果如下：



开发详细接口说明

1 对比同步接口

```
void compareSyncFile(QString leftPath, QString rightPath);
```

参数说明：leftPath 左边的文本文件路径，rightPath 右边的文本文件路径。

使用说明：

```
CCmp* pcmp = new CCmp(this);
//文件同步的调用方式，会弹出对比界面，可进行同步
pcmp->compareSyncFile(QString(".//testfile/1.txt"), QString(".//testfile/2.txt"));
```

运行后回弹出上图中的对比同步界面。

注意 pcmp 对象是新弹出窗口的父对象，如果 delete pcmp 会导致同步窗口也关闭。
最好是在需要的地方 CCmp* pcmp = new CCmp(this);
然后在主窗口关闭时去析构 pcmp。这部分是 qt 的知识了，我们假定您熟悉 QT/C++知识，不展开讲解。

2 对比接口

```
QObject* compareFile(QString leftPath, QString rightPath);
```

参数说明：leftPath 左边的文本文件路径，rightPath 右边的文本文件路径。
返回值：QObject* 是一个内部的对比对象，其 QT 父对象就是前面 new 出来的 pcmp，可以不手动释放，pcmp 析构时会自动回收。如果您使用一个 pcmp 对象，多次大量调用 compareFile 函数，则每次调用都会占用一个对比对象。为了减少内存开销，可以在对比处理结束后，手动 delete 释放一下返回值。

使用说明：

```
    CCmp* pcmp = new CCmp(this);
    //文件对比的方式，自己获取不同点后，进行自定义渲染显示。比较完成后，会发出 cmpResult 信号
    connect(pcmp, &CCmp::cmpFinished, this, &QtWidgetsApplication1::on_cmpSuccess);
    pcmp->compareFile(QString(".//testfile/1.txt"), QString(".//testfile/2.txt"));
```

用户需要自己处理 cmpFinished 信号，结果在该信号函数中。

2.1 signals cmpFinished 信号函数说明

```
    void cmpFinished(int resultType, QStringList* leftContents, QStringList* rightContents,
QVector<UnequalCharsPosInfo>* leftUnequalInfo, QVector<UnequalCharsPosInfo>*
```

```
rightUnequalInfo, const QList<BlockUserData*>* leftBlockData, const
QList<BlockUserData*>* rightBlockData);
```

参数说明：*resultType* 返回类型。0 表示对比完成 ;1 表示不需要对比，发生在有一个文件的实际内容是空。

QStringList* leftContents: 左边每一行的内容。
QStringList* rightContents：右边每一行的内容。

以上两个参数，返回了对比结果中每一行的内容，用户可以自己拿到每一行，然后进行显示。
注意 QString 里面是使用的 utf16 编码，输出时必须要使用 toutf8()接口，把 QString 内容转换为 utf8 编码的字符串进行展示。后面不同块的区间，都是按照 utf8 编码长度来进行度量的。务必要全部使用 utf8 编码。

QVector<UnequalCharsPosInfo>* leftUnequalInfo：左边内容的不同区间
QVector<UnequalCharsPosInfo>* rightUnequalInfo：右边内容的不同区间

前面说过每一行的内容在 leftContents 中，但是里面存在不同的部分，这部分用户要能够知道它们对应的位置。leftUnequalInfo 表示每一个不同块的区间。注意里面的区间是从整体内容而言的，不是单指某一行。

我们要先获取所有 utf8 内容的内容，通过如下方式：
```
QString leftText = leftContents->join("");
QByteArray leftUtf8Chars = leftText.toUtf8();
```
然后 leftUnequalInfo 的区间值，是表示的 leftUtf8Chars 中的范围。这里要注意转换一下。

如果想获取到某一行中，相对该行而言，行自身中的不同区间信息，可以参考例子中的 getUnEqualBlock 函数。该函数会对区间做相对转换，返回的是该行中不同块的区间。

const QList<BlockUserData*>* leftBlockData: 左边每一行对比结果的状态
const QList<BlockUserData*>* rightBlockData: 右边每一行对比结果的状态

每一行的状态说明，注意每行均存在一个 BlockUserData，说明该行对比的结果状态——是**相对行、不等行、还是对齐行**。BlockUserData 的定义在 blockuserdata.h 函数中。只需要关注其中的 int m_blockType 变量即可。行的状态如下枚举值 *BLOCKSTATUS* 所示。最后一个 TEMP_INSERT_BLOCK 对外不需要关注，内部使用。

```
enum BLOCKSTATUS {
    UNKNOWN_BLOCK = 0, //未知
    EQUAL_BLOCK = 1,//相等
    UNEQUAL_BLOCK, //不等
    PAD_BLOCK, //对齐
    LAST_PAD_EMPTY_BLOCK, // 最后一个用于对齐的空行，只在最后一行可能出现
    TEMP_INSERT_BLOCK //临时插入块
```

```
};
```
详细的使用方式，可以参考例子代码中的 on_cmpSuccess 函数。

参数释放说明：所有参数都不需要自己释放；框架会自己去处理释放；也不要修改函数的参数值。

3 接口类说明

```
typedef struct UnequalCharsPosInfo_ {
    int start; //不同块的开始偏移量
    int length; //不同块的长度
}UnequalCharsPosInfo;

class CC_EXPORT CCmp : public QObject
{
    Q_OBJECT
public:
    CCmp(QObject* parent=nullptr);
    virtual ~CCmp();

    void setCmpMode(int mode); //0 默认值,忽略行前和行尾的空白字符 1:只忽略行尾的空白字符 2:忽略行前、行中、行尾的所有空白字符。


void setCmpParameter(bool isBlankLineDoMatch, int lineMatchEqualRata);//对比参数。
//isBlankLineDoMatch 空行是否参与比较，默认值 true;
//lineMatchEqualRata 行认定为相等的相似率默认 50,越高匹配越严格，建议 50-90。

    //对比及同步文件。
    void compareSyncFile(QString leftPath, QString rightPath);

    //只对比文件，对比结果会以 cmpResult 的信号输出。对比结果是异步进行的。
    QObject* compareFile(QString leftPath, QString rightPath);

signals:
    void cmpFinished(int resultType, QStringList* leftContents, QStringList* rightContents, QVector<UnequalCharsPosInfo>* leftUnequalInfo, QVector<UnequalCharsPosInfo>* rightUnequalInfo, const QList<BlockUserData*>* leftBlockData, const QList<BlockUserData*>* rightBlockData); //对比完成后的信号，需要连接该信号，用户在自己的槽函数中自己渲染比较结果。只配合 compareFile 接口使用；compareSyncFile 不会触发该信号。
};
```
再次申明：本例子是在 qt5.12.10 环境下测试，动态库和 Lib 均基于 qt5.12.10 开发。
编译器是 vs2017。可以支持 linux 系统。如果您要非 qt 的环境的版本，请联系我们。**测试代码例子路径：**https://gitee.com/cxasm/cc-compare.git