POSIX正则表达式封装C接口说明

> 底层调用的接口为:
>
> > regcomp , regexec , regerror , regfree

源码

```
POSIXregexC.h
POSIXregexC.cpp
```

BRE与ERE区别

> POSIX 正则表达式分为：基本正则表达式和扩展正则表达式
>
> > 基本正则表达式(BRE:Basic Regular Expressions)
> > 扩展正则表达式(ERE:Extended Regular Expressions)

从查询的资料和接口测试的情况相接合来看规则如下：

BRE和ERE共有的规则为:

1. `.` ：匹配任意一个字符。
2. `[]` ：字符集匹配，匹配方括号中定义的字符集之一。
3. `[^]` ：字符集否定匹配，匹配没有在方括号中定义的字符。
4. `^` ：匹配开始位置。
5. `$` ：匹配结束位置。
6. `\` ：转义字符，用于转义特殊字符。

BRE和ERE都有的规则，但表达式不一样，BRE需要加转义字符，ERB不需要加转义字符
(例如表示匹配1次或多次,ERE为: `+` , BRE为: `\+`)

1. `?` ：最多一次匹配（零次或一次匹配）。
2. `+` ：至少一次匹配（一次或更多次匹配）。
3. `|` ：或运算，其左右操作数均可以为一个子表达式。
4. `()` ：定义子表达式。
5. `{}` ： `{m,n}` 至少 m 次，至多 n 次匹配； `{m}` 表示 m 次精确匹配； `{m,}` 表示至少 m 次匹配。

接口

> 类名: CPOSIXregC

1.通配符转正则表达式接口

```
    // 将通配符字符串转换为POSIX基本正则表达式(BRE:Basic Regular Expressions)字符串
    // return
    //   0  成功
    //   -1 分配内存失败
    static int glob_to_BRE(const char* glob, std::string &regStr) ;

//注意:此接口转换成正则表达式后会默认加在开头和结束符(^$),
//    例如: 通配字符中 *.cp 转换成正则表达式为 ^.*\.cp$
```

2.设置正则表达性属性接口

> 默认不设置时为 基本正则表达式(BRE)匹配

设置为扩展正则表达式(ERE) 匹配

```
//Whether the string str matches the regular expression
//return:
//   1  : match successful
//   0  : not match the regular expression
//   -1 : error
int isMatch(const char *str);
```

设置regcomp的其他标识

```
//regcomp(regex_t *preg, const char *regex, int cflags);'s cflags
//  cflags may be the bitwise-or of zero or more of the following:
//  REG_EXTENDED REG_ICASE REG_NOSUB REG_NEWLINE
void setCflags(int cflags);
```

3.设置某个正则字符串

```
//regcomp(regex_t *preg, const char *regex, int cflags);'s regex
//return:
//    0 : successful
//    -1: error
int setPattern(const char *regex);
```

4.判断某个字符串是否符合正则表达式

```
//Whether the string str matches the regular expression
//return:
//   1  : match successful
//   0  : not match the regular expression
//   -1 : error
int isMatch(const char *str);
```

5.获取错误信息

获取setPattern和isMatch接口的错误信息

```
//return mErrMsg
const char* getErrMsg(void);
```

使用举例

```
#include <stdio.h>
#include <string.h>

#include "POSIXregexC.h"


//测试POSIX正则表达式的匹配
void test_posix_reg( CPOSIXregC &tobj )
{
    int reti;
    char input[100];
    char pattern[100];

    // 得到正则表达式字符串(从用户输入)
    printf("Enter a regular expression pattern: ");
    fgets(pattern, sizeof(pattern), stdin);
    pattern[strcspn(pattern,"\n")]=0; //将最后的换行符替换为字符串结束符'\0'

    //设置正则表达式的： 正则字符串(含有正则字符的字符串)
    reti = tobj.setPattern( pattern );
    if ( reti != 0 ){
        fprintf(stderr, "Failed to compile the regular expression:errmsg[%s]\n",
                tobj.getErrMsg() );
        exit(1);

    }


    // 得到将要去匹配正则表达式的 字符串
    printf("Enter an input string: ");
    fgets(input, sizeof(input), stdin);
    input[strcspn(input,"\n")]=0; //将最后的换行符替换为字符串结束符'\0'

    //printf("input string:[%s] \n",input);

    //判断输入字符串是否符合正则表达式
    //return:
    //   1  : match successful
    //   0  : not match the regular expression
    //   -1 : error
    reti = tobj.isMatch( input );
    if (reti == 1) {
        //匹配成功
        printf("\n[%s] MATCH the regular expression [%s]\n",input, pattern);

    } else if (reti == 0) {
        //输入字符串不符合当前正则表达式
        //printf("[%s] NOT MATCH the regular expression [%s]\n",input, pattern);
        printf("\n%s\n",tobj.getErrMsg());

    } else {
        //错误
        fprintf(stderr, "\nRegular expression [%s] matching [%s] ERROR; errmsg=
[%s]\n",
                pattern,input,tobj.getErrMsg());
        exit(1);
    }
    return;
}
```

```
//测试 通配符字符串 转发成 POSIX正则表达式字符串
void test_glob_regex( void )
{
    int ret;
    std::string ostr;
    char gstr[100];

    // Get the wildcard string
    printf("Enter glob string: ");
    fgets(gstr, sizeof(gstr), stdin);
    gstr[strcspn(gstr,"\n")]=0; //将最后的换行符替换为字符串结束符'\0'

    ret = CPOSIXregC::glob_to_BRE( gstr, ostr );
    if ( ret != 0 ){
        fprintf(stderr, "POSIXregexC::glob_to_BRE return[%d]\n", ret );
        exit(1);
    }
    printf("\n wildcard string:[%s]\n", gstr);
    printf("\n POSIX regex string:[%s]\n", ostr.c_str());
    return;
}



int main(int argc, char* argv[]) {

    ////测试通配符转换到基本正则表达式
    //for ( int k=0; k<3; k++ ){
    //    test_glob_regex();
    //}
    //return 0;

    //1: 设置成支持 POSIX的 扩展正则表达式(ERE)
    //0: 默认不设置时 只支持 POSIX的 基本正则表达式(BRE)
    int ERE_flag=0;

    //for (int count = 0; count < argc; count++)
    //    printf("%d: %s \n", count, argv[count]);

    if ( argc > 1  && strcmp("1", argv[1]) == 0 ){
        ERE_flag=1;
    }

    printf("\nTIPS:POSIX正则表达式有两种匹配方式,对应此程序运行参数如下:\n");
    printf("\t1:不用操作参数时：POSIX的 基本正则表达式\n");
    printf("\t2:参数1时：扩展正则表达式(ERE) 匹配\n");

    CPOSIXregC tobj;

    if ( ERE_flag == 1 ) {
        //设置成支持 POSIX的 扩展正则表达式(ERE)
        tobj.setERE();
        printf("\n\t当前使用规则: POSIX的 扩展正则表达式(ERE) 匹配\n");
    }
    else {
        // 默认不设置时 只支持 POSIX的 基本正则表达式(BRE)
        printf("\n\t当前使用规则: POSIX的 基本正则表达式(BRE) 匹配\n");
    }
```

```
    for(int i=0; i<1; i++) {
        test_posix_reg( tobj );
    }

    return 0;
}
```