QTS单元测试框架

1.QTS单元测试框架介绍

目前QTS项目采用C/C++语言,而CppUnit就是xUnit家族中的一员,它是一个专门面向C++的单元测试框架。因此,QTS采用CppUnit测试框架是比较理想的选择。

CppUnit按照层次来管理测试,最底层的就是TestCase,当有了几个TestCase以后，可以将它们组织成TestFixture。在TestFixture中,可以建立被测试的类的实例,并编写TestCase对类实例进行测试,多个TestFixture可以通过TestSuite来对测试进行管理。

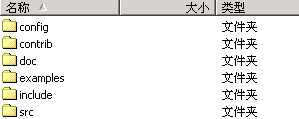
通过派生TestFixture类来设计某个类或某组相关功能的单元测试,Fixture定义公共函数setUp()初始化每个成员变量,tearDown()来释放setUp中使用的资源。在每个测试中,CPPUNIT\_ASSERT(bool)来判断某个函数和表达式的正确性,在派生类的声明中,通过CPPUNIT\_TEST来增加对应的测试函数,通过CPPUNIT\_TEST\_SUITE和CPPUNIT\_TEST\_SUITE\_END来分装所有的测试函数,规定这些测试函数执行的顺序。

2.QTS单元测试框架搭建

**2.1CppUnit 介绍**

**A、CppUnit源代码组成**

CppUnit测试框架的源代码可以到 http://sourceforge.net/projects/cppunit/ 上下载。下载解压后，你将看到如下文件夹：

  
图一

主要的文件夹有：

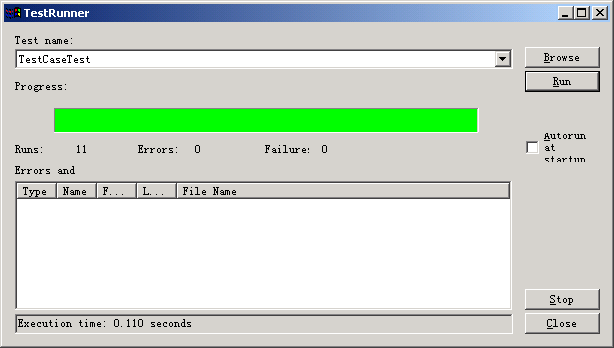
· doc: CppUnit的说明文档。另外，代码的根目录，还有三个说明文档，分别是INSTALL，INSTALL-unix，INSTALL-WIN32.txt。   
· examples: CpppUnit提供的例子，也是对CppUnit自身的测试，通过它可以学习如何使用CppUnit测试框架进行开发。   
· include: CppUnit头文件。

· src: CppUnit源代码目录。

B、**初识CppUnit测试环境**

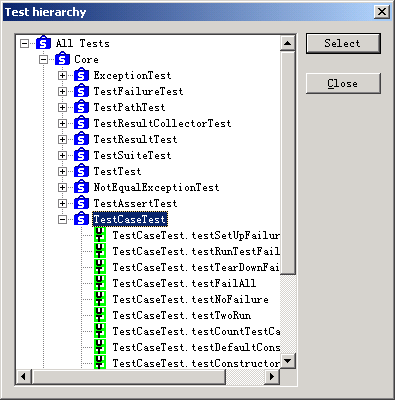
解压源代码包后， CppUnit结构如下:

    1、进入example文件夹，用VC打开examples.dsw。我们先来看看CppUnit自带的测试例子。这些例子都是针对CppUnit自身的单元测试集，一方面这是CppUnit作者开发CppUnit框架过程中写的测试用例，另一方面，我们可以通过这些例子来学习如何在我们自己的工程中添加测试用例。   
  
    2、将CppUnitTestApp工程设为Active Project（Win32 Debug），编译后运行，则可以看到CppUnit的基于GUI方式进行单元测试TestRunner的界面。点击“Run”，将会看到如图二所示界面：



图二

这是一个针对CppUnit的单元测试结果，它表明刚才我们做了11个测试，全部通过。   
点击“Browse”,我们还可以选择想要进行的单元测试，如图三：



图三

**2.2 CppUnit单元测试环境搭建**

**第一步:编译CppUnit 静态库文件\*.lib和动态库文件\*.dll:**

1. CppUnit的lib和dll

CppUnit为我们提供了两套框架库，一个为静态的lib，一个为动态的dll。

  cppunit project：静态lib

cppunit\_dll project：动态dll和lib

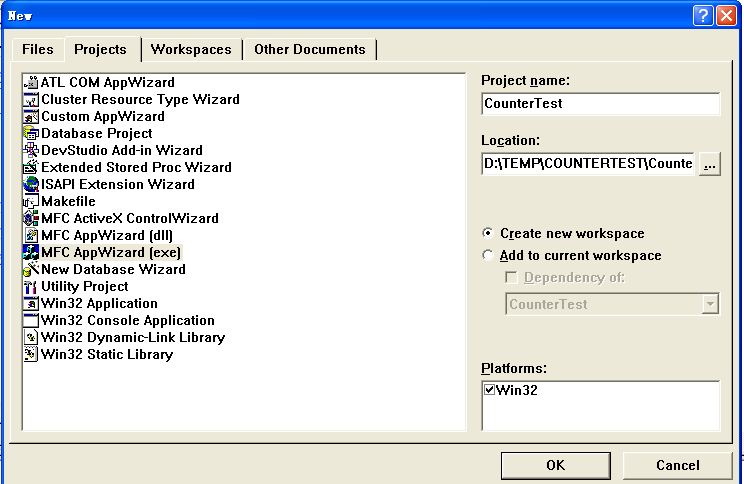
在开发中我们可以根据实际情况作出选择。进入src文件夹，打开CppUnitLibraries.dsw。分别编译这两个project，输出位置均为lib文件夹。

在开发中我们可以根据实际情况作出选择。进入src文件夹，打开CppUnitLibraries.dsw。在菜单上选择Build->Batch Build..->Rebuild All，输出位置均为lib文件夹。

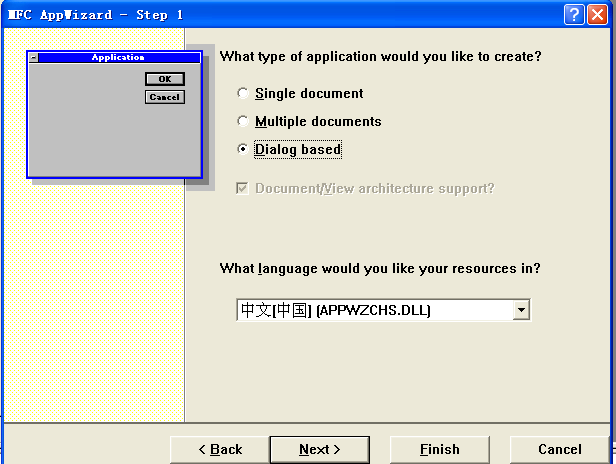
为了方便开发，我们把这些编译出来的lib和dll拷贝到我们自己建立的一个文件夹中(当然你也可以不这么做)，例如F:\Mytest\lib\，同时我们也把CppUnit源代码中include文件夹copy到我们自己的include文件夹下。

**第二步:建立基于对话框的工程**

打开VC，在File菜单项下选择New，建立基于dialog的工程。工程名Project name、存放位置Location可以自己决定，其他选项如下：



按OK确认后，进入如下界面。选择Dialog based选项，按Finish按钮后，一个空的基于对话框的工程就建立起来了。



**第三步:屏蔽工程对话框**

在工程CouterTest.cpp文件中（本指南中为该文件名，实际学习时根据自己的工程文件名而变），找到BOOL CCounterTestApp::InitInstance()方法，将如下蓝色代码注释掉：

BOOL CCounterTestApp::InitInstance()

{

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

/\*

CCounterTestDlg dlg;

m\_pMainWnd = &dlg;

int nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: Place code here to handle when the dialog is

// dismissed with OK

}

else if (nResponse == IDCANCEL)

{

// TODO: Place code here to handle when the dialog is

// dismissed with Cancel

}

\*/

// Since the dialog has been closed, return FALSE so that we exit the

// application, rather than start the application's message pump.

return FALSE;

}

由于我们希望这个Project运行后显示的是图2这样的CppUnit自带的界面，所以我们需要在Instance()中屏蔽掉原有的对话框（蓝色部分注释掉），代之以CppUnit的GUI。

**第四步:实现CppUnit测试执行器，并将测试套添加到测试执行器中。**

A、在BOOL CCounterTestApp::InitInstance()中，添加如下红色代码：

BOOL CCounterTestApp::InitInstance()

{

AfxEnableControlContainer();

// Standard initialization

// If you are not using these features and wish to reduce the size

// of your final executable, you should remove from the following

// the specific initialization routines you do not need.

#ifdef \_AFXDLL

Enable3dControls(); // Call this when using MFC in a shared DLL

#else

Enable3dControlsStatic(); // Call this when linking to MFC statically

#endif

//添加CppUnit的MFC类型的测试执行器

CppUnit::MfcUi::TestRunner runner;

//为被测试类（这里是CCounter）定义一个测试工厂(这里取名叫CounterTest):

CppUnit::TestFactoryRegistry &registry

= CppUnit::TestFactoryRegistry::getRegistry("CounterTest");

//并将工厂添加到测试执行器中

runner.addTest( registry.makeTest() );

//运行执行器，显示执行器GUI界面

runner.run();

/\*

CCounterTestDlg dlg;

m\_pMainWnd = &dlg;

int nResponse = dlg.DoModal();

if (nResponse == IDOK)

{

// TODO: Place code here to handle when the dialog is

// dismissed with OK

}

else if (nResponse == IDCANCEL)

{

// TODO: Place code here to handle when the dialog is

// dismissed with Cancel

}

\*/

// Since the dialog has been closed, return FALSE so that we exit the

// application, rather than start the application's message pump.

return FALSE;

}

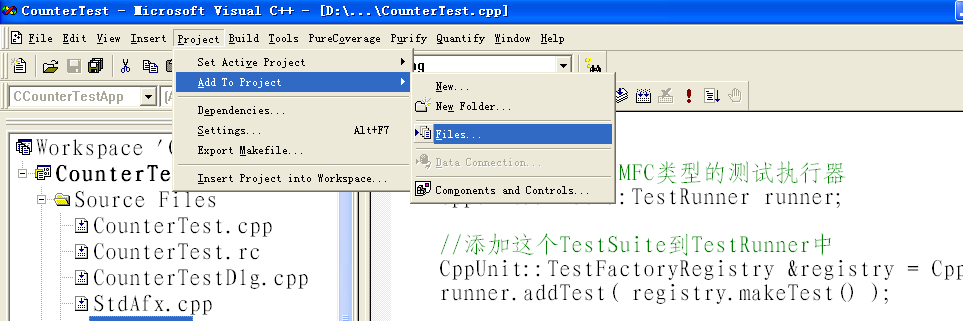
B、由于在BOOL CCounterTestApp::InitInstance()中引用了CppUnit的类，所以在文件开始处要添加如下头文件：

#include <cppunit/extensions/TestFactoryRegistry.h>

#include <cppunit/ui/mfc/TestRunner.h>

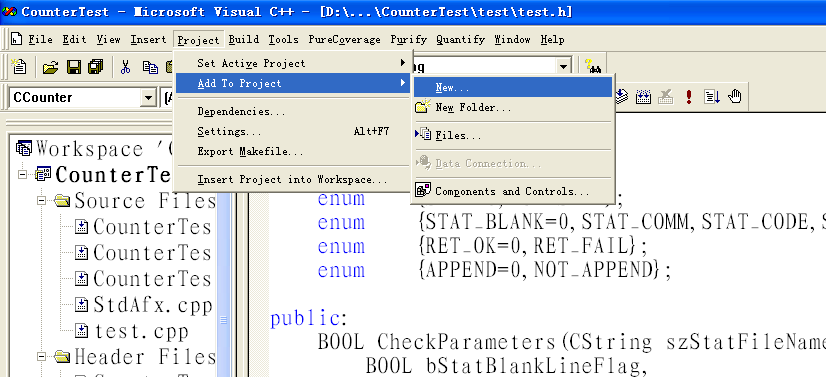
**第五步：添加被测对象CCounter。**

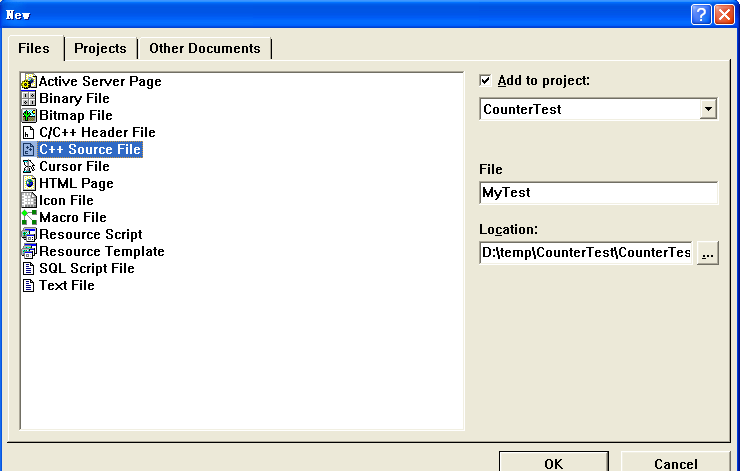
将被测对象所在文件（CounterMod.h和CounterMod.cpp） 添加到工程中：



**第六步：在工程中为被测对象CCounter编写测试类文件MyTest(可以自定义文件名)：**

按照下面示图加入测试类的\*.h文件和\*.cpp文件：





MyTest.h中的代码如下：

#include "cppunit/extensions/HelperMacros.h"

class IsCodeLineTest : public CppUnit::TestFixture {

// 声明一个TestSuite

CPPUNIT\_TEST\_SUITE( IsCodeLineTest);

// 添加测试用例到TestSuite, 定义新的测试用例需要在这儿声明一下

CPPUNIT\_TEST( Test1 );

// TestSuite声明完成

CPPUNIT\_TEST\_SUITE\_END();

public:

// 定义测试用例

void Test1 ();

};

MyTest.cpp中的代码如下（注意头文件要做相应的修改）：

#include "stdafx.h"

#include "MyTest.h"

#include "CounterMod.h"

// 把这个TestSuite注册到名字为"CounterTest"的工厂中

CPPUNIT\_TEST\_SUITE\_NAMED\_REGISTRATION( IsCodeLineTest,"CounterTest" );

#define RET\_OK 0

#define RET\_FAIL 1

void IsCodeLineTest::Test1()

{

//定义输入参数

int bIsComment;

CString szFileLine;

//定义期望输出

int iOkReturn;

int iOkIsComment;

//定义测试实际输出

int iResult;

CCounter m\_counter;

//用例输入

szFileLine = "int a";

bIsComment = false;

//期望输出

iOkReturn = RET\_OK;

iOkIsComment = false;

//驱动被测函数

iResult = m\_counter.IsCodeLine(szFileLine,bIsComment);

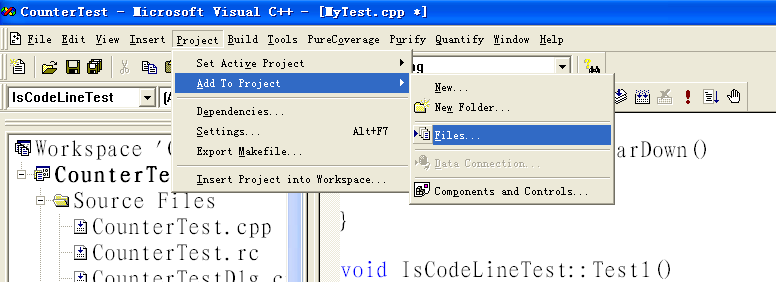
//结果比较

CPPUNIT\_ASSERT\_EQUAL(iOkReturn,iResult);

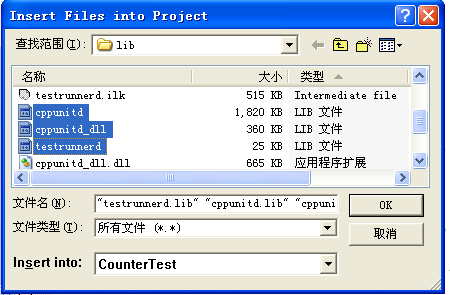
CPPUNIT\_ASSERT\_EQUAL(iOkIsComment,bIsComment);

}

**第七步：加入CppUnit 库文件：**

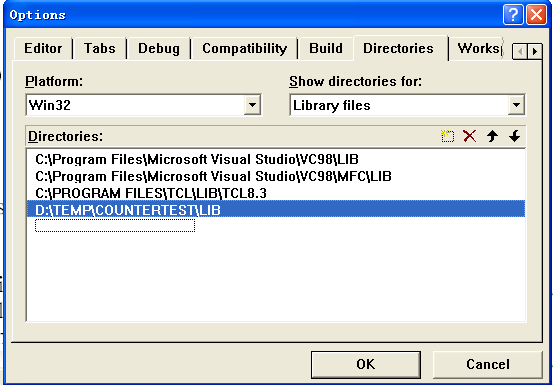


把CppUnit相关的lib文件和dll文件（cppunitd.lib,cppunitd\_dll.lib,testrunnerd.lib）加入到工程中：

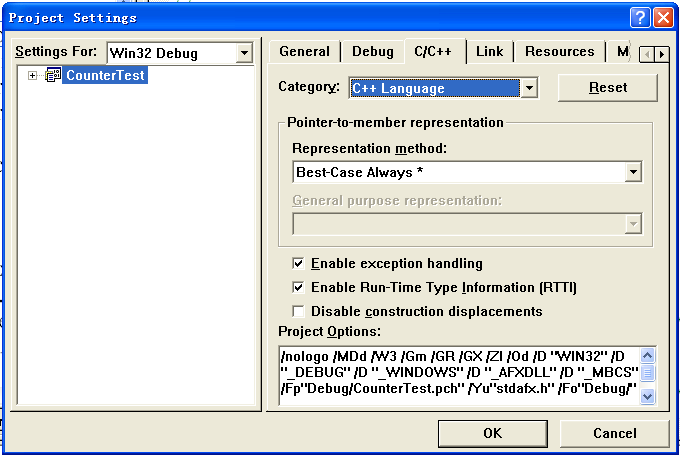


**第八步：设置头文件和lib库文件路径、打开RTTI开关、给dLL库设置环境变量：**

1. 在VC的tools/options/directories/include files和library files中设置CppUnit include文件路径和lib文件路径：



1. 在你的VC project中打开RTTI开关。具体位置Project Settings/C++/C++ Language：



  C、为TestRunnerd.dll设置环境变量   
　　 TestRunnerd.dll为我们提供了基于GUI的测试环境。为了让我们的测试程序能正确的调用它，需要把TestRunnerd.dll拷贝到你的工程路径下。或者最简单的方法是在操作系统的环境变量Path中添TestRunnerd.dll的路径，这样是最省事的。

**第九步：编译执行。编译连接成功后，运行测试，出现下面的界面,表示测试用例Test1运行成功.**

