

转 CppUnit的安装及使用指南

2014年05月11日 00:28:47 demystify 阅读数：3228 更多

转自：<http://wenku.baidu.com/view/a0a5867f27284b73f242508a.html>

第一部分 CppUnit安装

1、解压**cppunit-1.12.0.tar.gz**后进入**cppunit-1.12.0**目录。（用**winrar**解压即可）

2、在vc6中打开工作空间“**examples/examples.dsw**”，将**hostapp**设为**activeproject**，然后按**F7**构建。

3、选择vc6的菜单项“**build→batch build**”，点击对话框的“**build**”按钮。基本**ok**，除了最后的**simple_plugin**。

观察编译结果，若除**simple_plugin**外，还有其他错误，则按以下方式解决。

①错误提示：*Error spawning cl.exe*

在VC中点击“Tools”→“Option”→“Directories”，发现路径有误，重新设置“Executable Fils，Include Files，Library Files，Source Files”的路径。因此问题可以按照以下方法解决：打开vc界面点击VC“TOOLS（工具）”→“Option（选择）”→“Directories（目录）”重新设置“Executable Fils、Include Files、Library Files、Source Files”的路径。很多情况可能就一个盘符的不同（例如你的VC装在C，但是这些路径全部在D），改过来就OK了。（至少vc绿色版是这样，vc完全安装的情况下不会出现这种问题）

executatable files:

安装目录\Common\MSDev98\Bin

… 安装目录\VC98\BIN

安装目录\Common\TOOLS

安装目录\Common\TOOLS\WINNT

include files:

VC安装目录\VC98\INCLUDE

VC安装目录\VC98\MFC\INCLUDE

VC安装目录\VC98\ATL\INCLUDE

library files:

VC安装目录\VC98\LIB

VC安装目录\VC98\MFC\LIB

source files:

VC安装目录\VC98\MFC\SRC

VC安装目录\VC98\MFC\INCLUDE

VC安装目录\VC98\ATL\INCLUDE

VC安装目录 | VC98\CRT\SRC

②错误提示: *Fatal error LINK1104: cannot openfile "mfc42u.lib"*

网上下载mfc42u.lib、mfc42ud.lib即可。

③错误提示: *Error executing c:\windows\system32\cmd.exe*

该错误可以忽略。

4、打开src/cppunitlibraries.dsw后进行"batchbuild"。(若testrunnerdsplugind.dll构建失败, 是因为testrunnerdsplugin.dll已经注册。)

5、打开examples/examples.dsw后将cppunittestapp设为activeproject, 然后运行ok。

6、选择vc6的菜单项"tools> customize..."。选择"customize"对话框的"add-insand macro files"标签页。点击其中的"browse..."按钮, 双击lib/testrunnerdsplugin.dll。(注册这个插件后, 在随后的测试过程中, 若出现错误, 则双击错误后可以在vc6中打开相应行。)

7、选择vc6的菜单项"tools> options..."。选择"options"对话框的"directories"标签页。

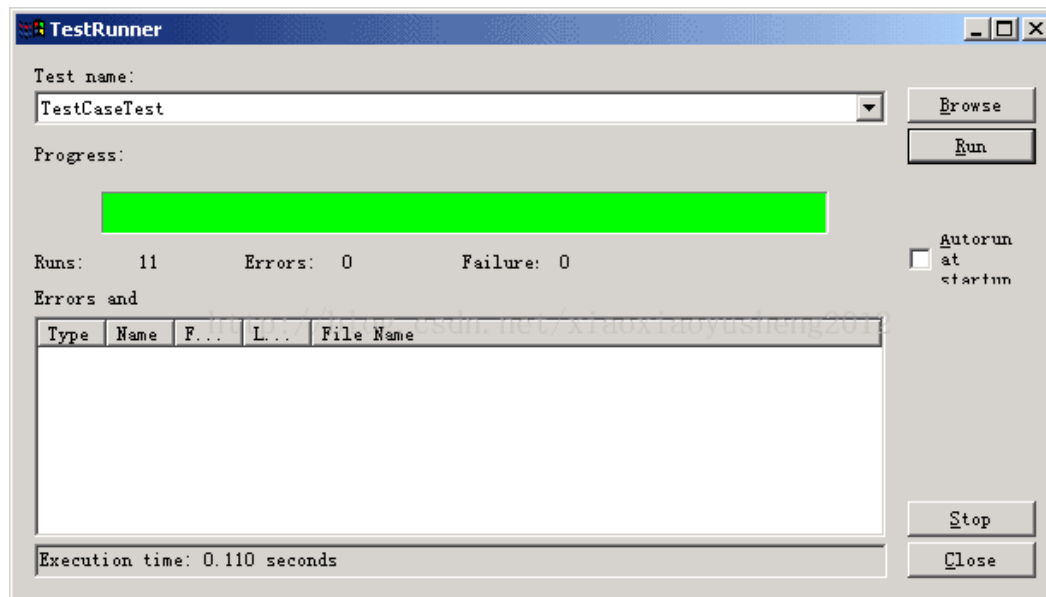
- 选择"include files", 添加新项"cppunit-1.12.0目录\include"。

- 选择"library files"，添加新项"cppunit-1.12.0目录\lib"。
- 选择"source files"，添加新项"cppunit-1.12.0目录\src\cppunit"。

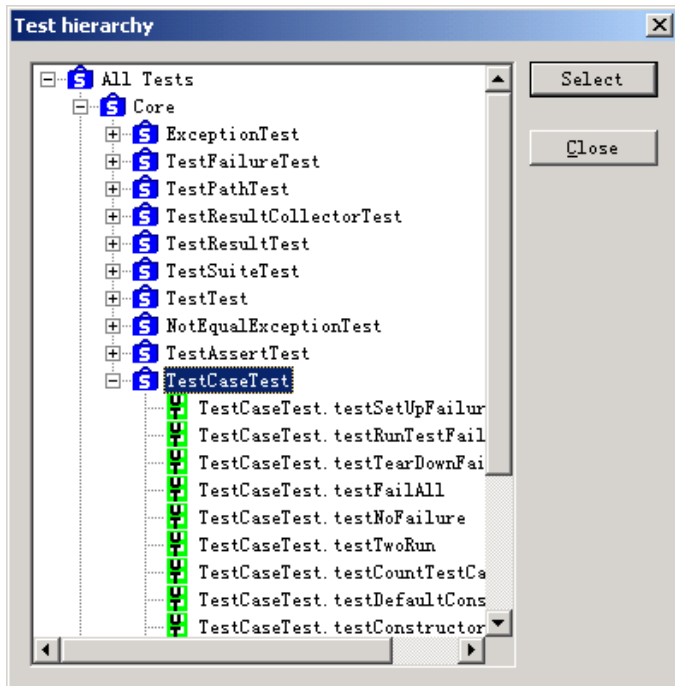
第二部分初识CppUnit

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ActiveProject（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"，将会看到如下界面，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。



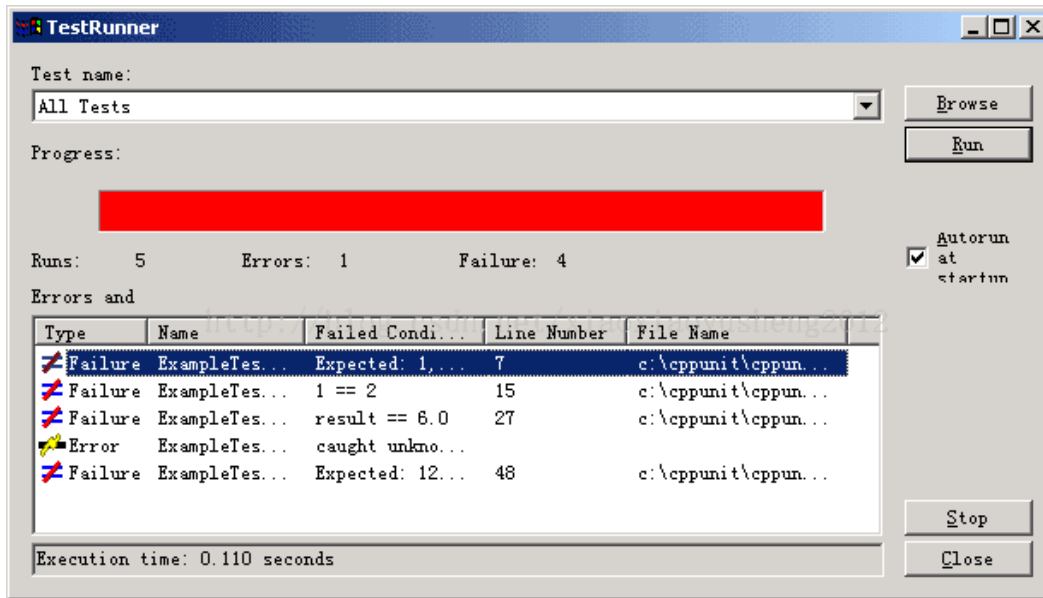
点击"Browse",我们还可以选择想要进行的单元测试。CppUnit将所有的单元测试按照树的结构来表示。在CppUnit中，最小的测试单元，称为TestMethod（测试方法），而多个相关的测试方法又可以组成一个TestCase测试用例。多个测试用例又组成TestSuite测试包。测试包互相嵌套在一起，就形成了上面我们看到的树结构。我们可以选择其中任意的树节点来进行单元测试。



3、将CppUnitTestMain工程设置为ActiveProject (Win32 Debug) , 编译并运行, 得到如下界面, 这是一个基于文本方式的单元测试环境。



4、将HostApp工程设置为ActiveProject (Win32 Debug) , 编译运行, 得到如下界面, 这亦是一个对CppUnit自身进行的测试, 只不过它向我们演示的是各种失败的测试。在基于GUI的测试环境中, 若测试不成功, 进度条显示红色, 反之则为绿色。从测试结果我们可以看到失败的单元测试名称, 引起测试不能通过的原因, 以及测试失败的语句所在的文件及所在行数。



第三部分 CppUnit原理

在 CppUnit中，一个或一组测试用例的测试对象被称为*Fixture*。*Fixture*就是被测试的目标，可能是一个对象或者一组相关的对象，甚至一个函数。

通常写一个 *TestCase* 的步骤包括：

- ①初始化操作，比如：生成一组被测试的对象，初始化值；
- ②按照要测试的某个功能或者某个流程对*fixture*进行操作；
- ③验证结果是否正确；
- ④ 清理工作，比如对*fixture*的及其他的资源进行释放等。

对 *fixture* 的多个测试用例，通常(1)、(4)部分代码都是相似的，CppUnit在很多地方引入了*setUp*和*tearDown*虚函数。可以在*setUp*函数里完成(1)初始化代码，而在 *tearDown*函数中完成(4)代码。具体测试用例函数中只需要完成(2)、(3)部分代码即可，运行时CppUnit会自动为每个测试用例函数运行 *setUp*，之后运行 *tearDown*，这样测试用例之间就没有交叉影响。

对 *fixture* 的所有测试用例可以被封装在一个 *CppUnit::TestFixture* 的子类（命名惯例是[ClassName]Test）中。然后定义这个*fixture*的*setUp*和 *tearDown*函数，为每个测试用例定义一个测试函数（命名惯例是 *testXXX*）。

```
class MathTest : public CppUnit::TestFixture {
```

protected:

*int*m_value1,m_value2;

public:

MathTest() {}

//步骤（1）初始化函数

*void*setUp () {

m_value1 =2;

m_value2 =3;

}

//测试加法的测试函数

*void*testAdd () {

//步骤（2）对 *fixture* 进行操作

*int*result =m_value1 +m_value2;

//步骤（3）验证结果是否争取

CPPUNIT_ASSERT(result ==5);

}

//步骤（4）没有什么清理工作*tearDown*.

*void*teardown() {}

}

在测试函数中对执行结果的验证成功或者失败直接反应这个测试用例的成功和失败。*CppUnit*提供了多种验证成功失败的方式:

```
CPPUNIT_ASSERT(condition)           //确信condition为真

CPPUNIT_ASSERT_MESSAGE(message, condition)  //当condition为假时失败,并打印message

CPPUNIT_FAIL(message)                //当前测试失败,并打印message

CPPUNIT_ASSERT_EQUAL(expected, actual)     //确信两者相等

CPPUNIT_ASSERT_EQUAL_MESSAGE(message, expected, actual)  //失败的同时打印message

CPPUNIT_ASSERT_DOUBLES_EQUAL(expected, actual, delta) //当expected和actual之间差大于delta时失败
```

要把对 *fixture* 的一个测试函数转变成一个测试用例, 需要生成一个 *CppUnit::TestCaller* 对象。而最终运行整个应用程序的测试代码的时候, 可能需要同时运行对一个 *fixture* 的多个测试函数, 甚至多个 *fixture* 的测试用例。

CppUnit 中把这种同时运行的测试案例的集合称为 *TestSuite*。而 *TestRunner* 则运行测试用例或者 *TestSuite*, 具体管理所有测试用例的生命周期。目前提供了 3 类 *TestRunner*, 包括:

```
CppUnit::TextUi::TestRunner  // 文本方式的TestRunner

CppUnit::QtUi::TestRunner    // QT方式的TestRunner

CppUnit::MfcUi::TestRunner   // MFC方式的TestRunner
```

下面是一个 *TestRunner* 的例子:

```
CppUnit::TextUi::TestRunnerrunner;

CppUnit::TestSuite *suite=new CppUnit::TestSuite();

//添加一个测试用例

suite->addTest(new CppUnit::TestCaller<MathTest> ("testAdd", testAdd));

//指定运行TestSuite

runner.addTest(suite );

//开始运行,自动显示测试进度和测试结果
```

```
runner.run("",true); //Run all tests and wait
```

第四部分模拟测试代码编写

（一）简单方法

1、新建Win32 Console Application工程，命名为“dd”（可取其它名字），并指定存储路径：

2、在测试项目中设置（非常重要）

①在VC中，在菜单中，选择Project→Settings→'C/C++'→Category列表中选择'C++ Language'，选择'enableRun-Time Type Information (RTTI)'。

②在VC中，在菜单中，选择Project→Settings→'C/C++'→Category列表中选择'Codegeneration'，在Use run-time library列表中，对于Debug版，选择'DebugMultithreaded DLL'，对于release版，选择'MultithreadedDLL'

；

③在VC中，在菜单中，选择Project→Settings→'Link'，在'Object/librarymodules'中添入需要的lib文件：cppunitX.lib（debug模式为cppunitd.lib,release模式为cppunit.lib）、testrunnerX.lib（debug模式为testrunnerd.lib, release模式为testrunner.lib, debug Unicode模式为testrunnerud.lib, release Unicode模式为testrunneru.lib）

3、新建测试类“MathTest”

4、修改文件“MathTest.h”

```
// MathTest.h关键代码
#include "cppunit/extensions/HelperMacros.h"
class MathTest : public CppUnit::TestFixture
{
public:
    MathTest();
    virtual ~MathTest();

protected:
    int m_value1, m_value2;

public:
    void setUp();           //初始化函数
    void tearDown();        //清理函数
    void testAdd ();        //测试加法的测试函数
    void testMinus();       //测试减法的测试函数，可以添加新的测试函数
};
```

5、修改文件“MathTest.cpp”


```
//MathTest.cpp关键代码
void MathTest::setUp()
{
    m_value1 = 2;
    m_value2 = 3;
}

void MathTest::tearDown()
{
}

void MathTest::testAdd()
{
    int result = m_value1 + m_value2;
    CPPUNIT_ASSERT( result == 5 );
}

void MathTest::testMinus()
{
    int result = m_value1 - m_value2;
    CPPUNIT_ASSERT( result == -1 );
}
```

6、修改main函数

```
//main
#include "stdafx.h"
#include "mathTest.h"
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>

int main(int argc, char* argv[])
{
    CppUnit::TextUi::TestRunner runner;
    CppUnit::TestSuite *suite= new CppUnit::TestSuite();
    // 添加一个测试用例
    suite->addTest(new CppUnit::TestCaller<MathTest> ("testAdd",&MathTest::testAdd));

    suite->addTest(new CppUnit::TestCaller<MathTest> ("testMinus", &MathTest::testMinus));
    // 指定运行TestSuite
    runner.addTest( suite );
    // 开始运行, 自动显示测试进度和测试结果
    runner.run( "", true );    // Run all tests and wait

    return 0;
}
```

7、编译, 若编译出错, 请验证是否按照第2步进行了相应设置。以下解决方式仅作参考。

①错误提示: *error C2039: 'TestCaller' : is nota member of 'CppUnit'*

该错误对应的错误代码行为: “*suite->addTest(new CppUnit::TestCaller<MathTest> ("testAdd", &MathTest::testAdd));*”, 可能是以下原因造成:

a. 'TestCaller'未能识别MathTest, 请查看是否包含头文件: #include mathtest.h

b. 类MathTest未继承自CppUnit::TestFixture

②错误提示: *error LNK2001: unresolved externalsymbol "public: virtual __thiscall CppUnit:: TextTestRunner::~~TextTestRunner(void)" (??1TextTestRunner@CppUnit@@@UAE@XZ)*

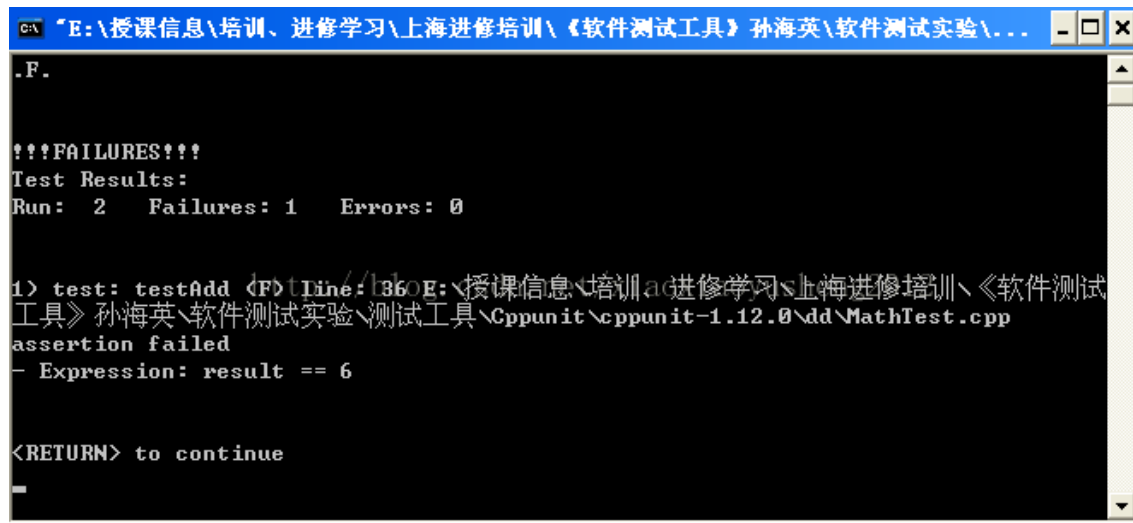
选择vc6的菜单栏*Project→Settings→Link'*, 添加需要的lib文件: *cppunitd.lib、testrunnerd.lib*

③错误提示: *error C2653: 'CppUnit' : is not aclass or namespace name*

该错误对应的错误代码行为: “*class MathTest : public CppUnit::TestFixture*”, 是因为未包含头文件: *#include"cppunit/extensions/HelperMacros.h"*

8、运行, 以下是测试成功界面及测试失败界面。



A screenshot of a Windows command prompt window. The title bar shows the path "E:\授课信息\培训、进修学习\上海进修培训\《软件测试工具》孙海英\软件测试实验\...". The window has a black background with white text. It displays the output of a CppUnit test run. At the top, it says ".F.". Below that, it says "!!!FAILURES!!!". Then, it shows "Test Results:" followed by "Run: 2 Failures: 1 Errors: 0". The next line is "1) test: testAdd (F) Line: 36". The following line is the full path to the test file: "E:\授课信息\培训、进修学习\上海进修培训\《软件测试工具》孙海英\软件测试实验\测试工具\Cppunit\cppunit-1.12.0\dd\MathTest.cpp". Below that, it says "assertion failed" and then "- Expression: result == 6". At the bottom, it says "<RETURN> to continue".

```
.F.

!!!FAILURES!!!
Test Results:
Run: 2   Failures: 1   Errors: 0

1) test: testAdd (F) Line: 36 E:\授课信息\培训、进修学习\上海进修培训\《软件测试工具》孙海英\软件测试实验\测试工具\Cppunit\cppunit-1.12.0\dd\MathTest.cpp
assertion failed
- Expression: result == 6

<RETURN> to continue
```

(二) 使用 *TestSuite*

按照上面的方式，如果要添加新的测试用例，需要把每个测试用例添加到 *TestSuite* 中，而且添加新的 *TestFixture* 需要把相对应所有头文件（即每一个测试集合定义所在的头文件）添加到 *main* 函数所在文件中，比较麻烦。为此 *CppUnit* 提供了 *CppUnit::TestSuiteBuilder*，*CppUnit::TestFixtureRegistry* 和一堆宏，用来方便地把 *TestFixture* 和测试用例注册到 *TestSuite* 中。

步骤同上，只是一些细节问题，以图示方式描述，请对照着来看。

```
// MathTest.h关键代码
#include "cppunit/extensions/HelperMacros.h"
class MathTest : public CppUnit::TestFixture
{
    CPPUNIT_TEST_SUITE(MathTest); // 声明一个TestSuite, 将该类作为测试用例集合, 注意, 与类名一致, 否则出错
    CPPUNIT_TEST(testAdd);       // 添加测试用例到TestSuite
    CPPUNIT_TEST(testMinus);     // 继续添加测试用例...
    CPPUNIT_TEST_SUITE_END();    // TestSuite声明完成

public:
    MathTest();
    virtual ~MathTest();

protected:
    int m_value1, m_value2;

public:
    void setUp(); // 初始化函数
    void tearDown(); // 清理函数
    void testAdd (); // 测试加法的测试函数
    void testMinus(); // 测试减法的测试函数, 可以添加新的测试函数
};
```

<http://blog.csdn.net/xiaoxiaoyusheng2012>

//MathTest.cpp关键代码

// 把这个TestSuite注册到名字为"alltest"的TestSuite中, 如果没有定义会自动定义
// 也可以CPPUNIT_TEST_SUITE_REGISTRATION(MathTest); 注册到全局的一个未命名的TestSuite中.
CPPUNIT_TEST_SUITE_NAMED_REGISTRATION(MathTest, "alltest");

void MathTest::setUp()

```
{  
    m_value1 = 2;  
    m_value2 = 3;  
}
```

void MathTest::tearDown()

```
{  
  
}
```

<http://blog.csdn.net/xiaoxiaoyusheng2012>

void MathTest::testAdd()

```
{  
    int result = m_value1 + m_value2;  
    CPPUNIT_ASSERT( result == 5 );  
}
```

void MathTest::testMinus()

```
{  
    int result = m_value1 - m_value2;  
    CPPUNIT_ASSERT( result == -1 );  
}
```

```
//main
#include "stdafx.h"
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>

int main(int argc, char* argv[])
{
    CppUnit::TextUi::TestRunner runner;

    // 从注册的TestSuite中获取特定的TestSuite, 没有参数获取未命名的TestSuite.
    CppUnit::TestFactoryRegistry &registry = CppUnit::TestFactoryRegistry::getRegistry("alltest");

    //若没有注册名字, 在MathTest.cpp文件中只是调用了CPPUNIT_TEST_SUITE_REGISTRATION( MathTest );
    //CppUnit::TestFactoryRegistry &registry = CppUnit::TestFactoryRegistry::getRegistry();

    // 添加这个TestSuite到TestRunner中
    runner.addTest( registry.makeTest() );
    // 运行测试
    runner.run();

    return 0;
}
```

（三）可视化方式

- 1、新建基于对话框的应用程序工程，命名为“gg”（可取其它名字），并指定存储路径；
- 2、在测试项目中进行相应设置（同上）
- 3、将testrunnerd.dll拷贝至工程目录下（非常重要，TestRunner.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，TestRunner.dll必须位于你的测试程序的路径下。）
- 4、修改InitInstance所在文件内容，替换为采用CppUnit的测试对话框，随后的测试过程将以图形方式显示。

```
//包含头文件
#include <cppunit/ui/mfc/TestRunner.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
```

```
//InitInstance函数体关键代码
CppUnit::MfcUi::TestRunner runner; //MFC方式的TestRunner, 即图形界面显示
runner.addTest( CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest() );
runner.run(); //show UI

/* CGgDlg 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
}
*/
```

5、新建测试类CplusTest

6、修改PlusTest.h

```
//PlusTest.h关键代码
class CPlusTest : public CppUnit::TestFixture
{
    CPPUNIT_TEST_SUITE(CPlusTest);
    CPPUNIT_TEST(testAdd);
    CPPUNIT_TEST_SUITE_END();

public:
    void testAdd();
    CPlusTest();
    virtual ~CPlusTest();
};
```

7、修改PlusTest.cpp

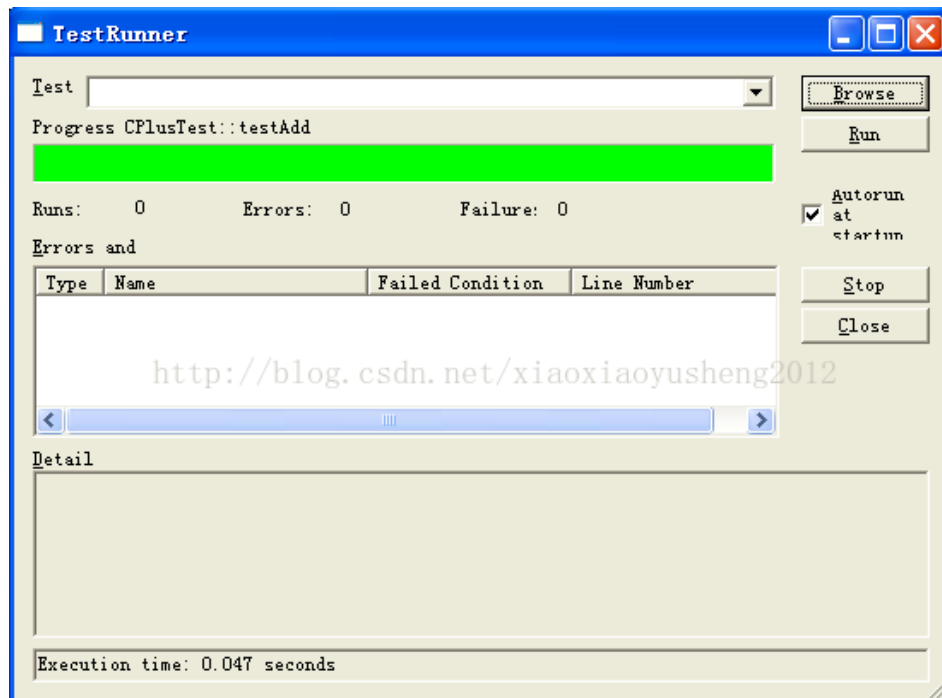
```
//PlusTest.cpp关键代码
#include "myplus.h" //被测类的头文件
CPPUNIT_TEST_SUITE_REGISTRATION(CPlusTest);
void CPlusTest::testAdd()
{
    CMyPlus plus;
    CPPUNIT_ASSERT(plus.Add(2,3)==5);
}
```

8、新建被测类“CPlusTest”（可参考“第五部分”）

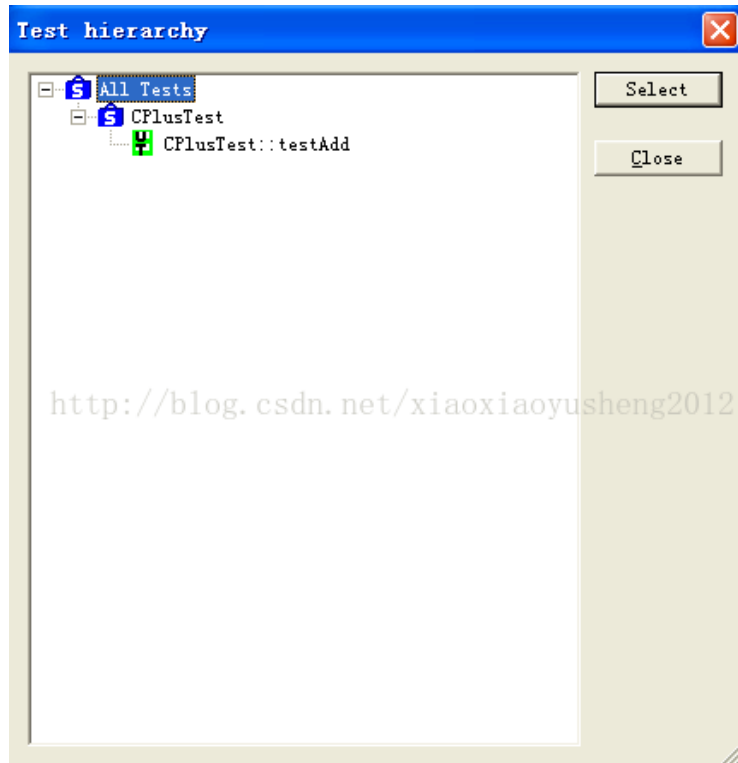
```
class CMyPlus
{
public:
    int Add(int x,int y);
    CMyPlus();
    virtual ~CMyPlus();
};

int CMyPlus::Add(int x,int y)
{
    return x+y;
}
```

9、编译运行，即可出现图形化界面。



点击Browse，可看到在测试代码编写过程中所设计的测试集合框架。



选定测试集合后，回到`TestRunner`对话框，点击`Run`，即可开始测试。

第五部分实际测试代码编写

1、新建`Win32 Console Application`工程，命名为“`ee`”（可取其它名字），并指定存储路径；

2、在测试项目中进行相应设置（同“第四部分”）

3、新建测试类`MyTest`

4、修改文件`MyTest.h`”

```
//MyTest.h关键代码
#include "cppunit/extensions/HelperMacros.h"
class MyTest : public CppUnit::TestFixture
{
public:
    void setUp(void){}
    void tearDown(void){}
    void testAdd(void); //实际的测试方法

public:
    MyTest();
    virtual ~MyTest();
};
```

5、修改文件MyTest.cpp

```
//MyTest.cpp关键代码
#include "MyPlus.h" //被测试类的头文件
void MyTest::testAdd() //实际的测试方法的实现
{
    CMyPlus plus; //CMyPlus是被测试类
    CPPUNIT_ASSERT(plus.Add(1,2) == 3); //Add是被测试函数
}
```

6、修改main函数

```
//main
#include "stdafx.h"
#include "mytest.h"
#include <cppunit/ui/text/TestRunner.h>
#include <cppunit/extensions/TestFactoryRegistry.h>

int main(int argc, char* argv[])
{
    // Add the top suite to the test runner
    CppUnit::TextUi::TestRunner runner;
    CppUnit::TestSuite *suite= new CppUnit::TestSuite();
    // 添加一个测试用例
    suite->addTest(new CppUnit::TestCaller<MyTest> ("testAdd",&MyTest::testAdd));

    // 指定运行TestSuite
    runner.addTest( suite );
    // 开始运行，自动显示测试进度和测试结果
    runner.run( "", true ); // Run all tests and wait

    return 0;
}
```

7、添加被测类CMyPlus

```
//MyPlus.h关键代码
class CMyPlus
{
public:
    int Add(int x,int y);
    CMyPlus();
    virtual ~CMyPlus();
};

//MyPlus.cpp关键代码
int CMyPlus::Add(int x, int y)
{
    return x+y;
}
```

8、编译运行。



威图的机柜

机柜上一个u是几个格口

想对作者说点什么？

我来说两句

CppUnit的安装配置与使用



631

1、将cppunit-1.12.1(FULL).rar解压到C盘（一般情况下，需要与VC++6.0在同一张逻辑盘上）v注册testr...

Linux下安装cppunit



2063

cppunit的介绍页面在这里，源码下载在这里 下载源码包：从这里选择一个版本的cppunit下载，目前的版本是1.12.1 注意：不要在sourceforge-cppunit主...

月薪40K+！区块链以太坊开发八周学会

区块链DApp开发学习路线图，月薪4万很轻松