**C++Test代码静态分析**

严庆兵 2013-2-25

1. **工具介绍**

C++ Test是Parasoft公司出品的一个针对C/C++源代码进行自动化审查和单元测试的工具。它可以对源代码进行三种测试：白盒测试、黑盒测试以及回归功能测试。对C++代码进行静态分析和代码覆盖，并检查编码规范。对C++类自动生成测试用例、驱动模块/桩模块。

对于程序员来说可以使用代码静态分析、代码审查、运行期bug分析、自动白盒测试功能。

1. **安装**

C++Test有独立版和插件版(内嵌到VS开发环境版本和独立IDE版本(还支持Linux\Unix等系统))。整理了一个破解版本，如下：

1. 插件版

下载地址:

传到skydrive中了,试试能下载吗.

https://skydrive.live.com/?cid=CB913F42229744B1&id=CB913F42229744B1!358

独立版本破解:

把lic\_client.jar复制到

c:\Program Files (x86)\Parasoft\Test\9.0\plugins\com.parasoft.xtest.libs\_9.0.0.20100729\Parasoft\

这个目录下覆盖原始文件.

运行后随便输入几个注册码 可以成功。

插件版本破解：

把plugins-c++Test For Visual Studio.7z中的文件覆盖到

c:\Program Files (x86)\Parasoft\C++test for Visual Studio\9.2\plugins\ 这个目录中

把plugins-Test for Virsual Studio.7z 中的文件覆盖到

c:\Program Files (x86)\Parasoft\Test for Visual Studio\9.2\plugins这个目录中

运行ide随便输如几个注册码 可以成功

concerto破解:

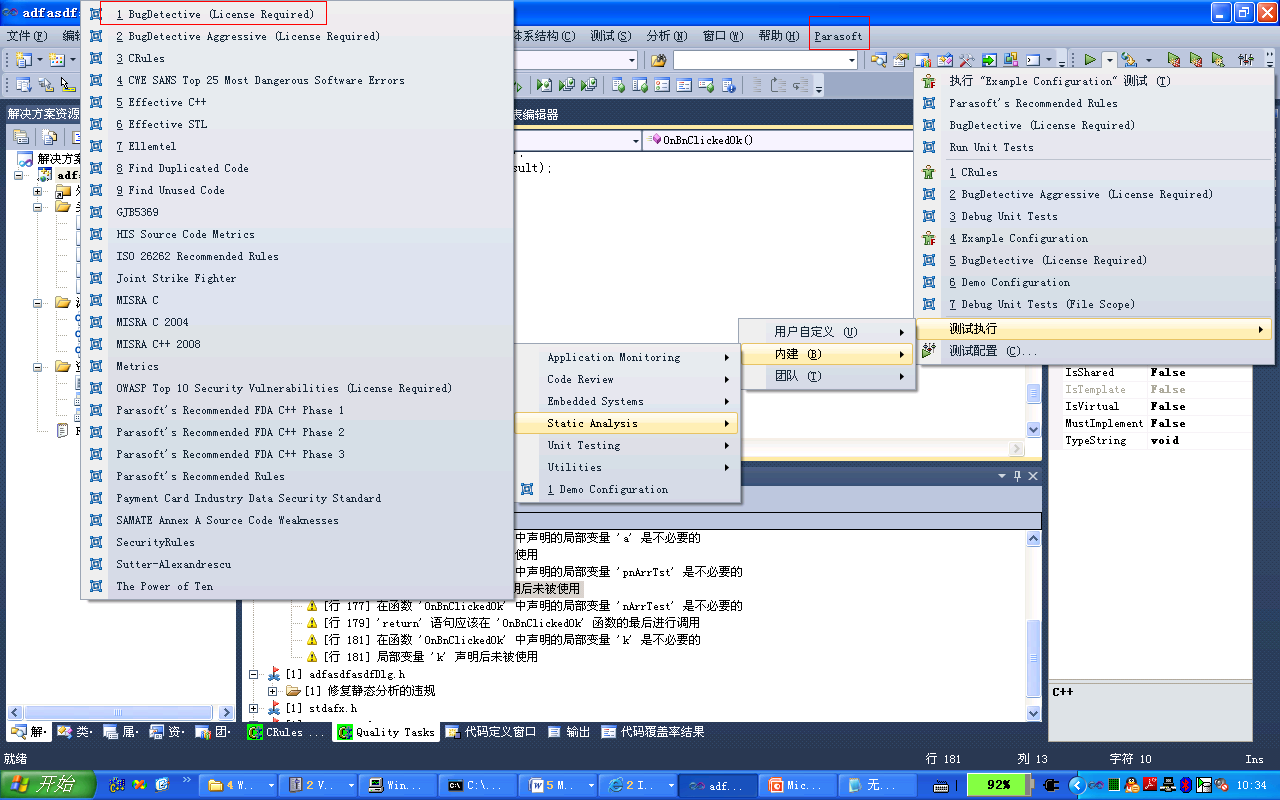
安装后把xtest.lic.client-for concerto.jar改名为:xtest.lic.client.jar 然后覆盖到tomcat/lib(好像是这个目录,可以搜索下这个文件). 运行浏览器,打开网页,随便输入key就可以.

B．破解独立版本

Parasoft C Plus Plus Test v6.0.0.5 英文版(从CSDN下载)，安装后，将toolkit.dll文件替换到安装目录，即破解。

1. **静态分析和代码审查**
2. **静态分析**

安装C++Test插件版本后，打开VS2010，创建或打开一个新工程，如下图操作：



将执行C++Test默认的静态检测，默认方式包含了常用的代码审查、静态检测BUG。例如对以下一段有问题的代码进行静态分析：

int Calc(void\* pTest, int& nK)

{

int\* pnTest = (int\*)pTest;

\*pnTest = 123;

nK = 100;

return (\*pnTest) \* nK;

}

int GetVal(int nPara)

{

return nPara - 10;

}

void CaaaDlg::OnBnClickedOk()

{

//例子1：设置了匈牙利命名编码规则

{

//采用匈牙利命名变量，整型应该以"i"字母打头, 下面代码应该是"iAbc"

int abc=123;

}

//例子2：if下面单条语句的花括号检查规则

{

int i=0,k=0;

if (0==i) //'if' 后面应该跟着块语句 ("{"符号)

k = 0;

else

k = 1;

}

//例子3：动态计算检查除数为零

{

int a =100 / GetVal(10); //GetVal函数计算后返回的是0， 除数为0，静态检查中将能发现

}

//例子4：数组越界

{

int nArrAbc[500];

nArrAbc[501] = 123;

int iIndex=100;

nArrAbc[iIndex+499] = 123;

}

//例子5：内存忘记释放

{

int\* pT = new int[512];

}

//例子6：根据实际需要，用户自定义的新规则

{

int i=0, j=2, k=0;

if (i < j){ //定义了自己的规则，当有"if (a < b)"形式时，例如静态分析显示"看看测试出问题后，是否输出这句话.YanQingbing"

k = 2;

}

else{

k =3;

}

}

//例子7：白盒测试：对一个函数自动生成测试用例，能自动测试

{

int abc=100;

int k=123;

int result = Calc(&abc, k);

TRACE(\_T("result[%d]"),result);

}

//例子8：函数return后面的代码，都是废代码，静态分析提示

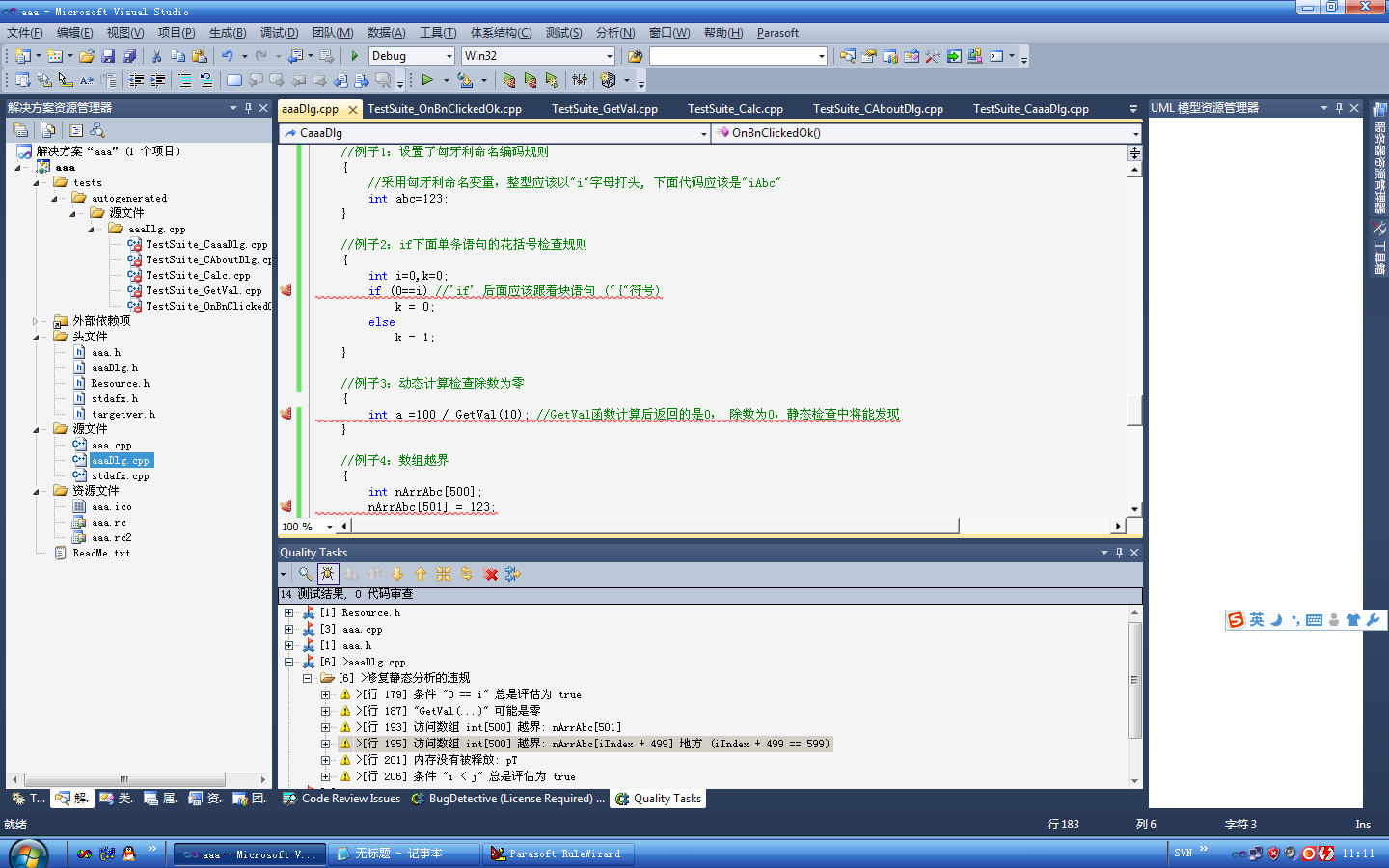
return;

int iBack=0;

CDialogEx::OnOK();

}

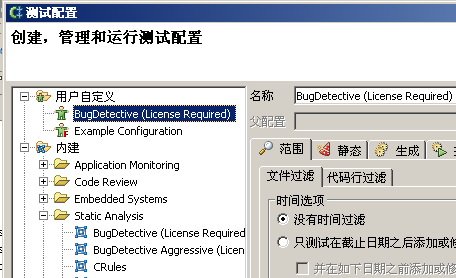
静态分析后，结果如下：



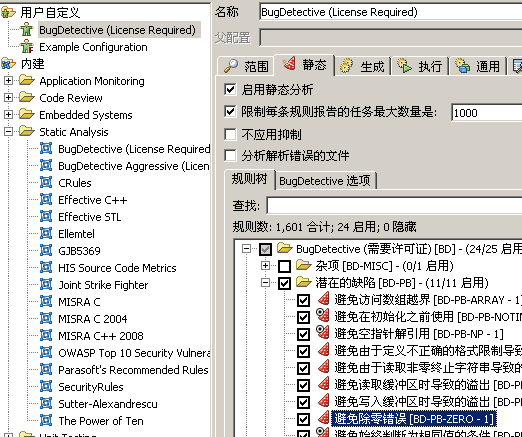
1. **静态分析环境设置和添加自定义规则**
2. **静态分析 —— BugDetective**
3. 点击“C++test → 测试配置” ，展开“内建 → Static Analysis” ，然后Right click on“BugDetective（License Required）” ，选择“复制” ：



1. 点击复制之后，可以在“用户自定义”列表下看见一个“BugDetective（License Required）” （为什么要做这步操作呢？其实在“内建”中，也是可以执行这个测试的，但是不可以修改BugDetective的配置了。只有复制到“用户自定义”列表下后，配置才是可修改状态的）：

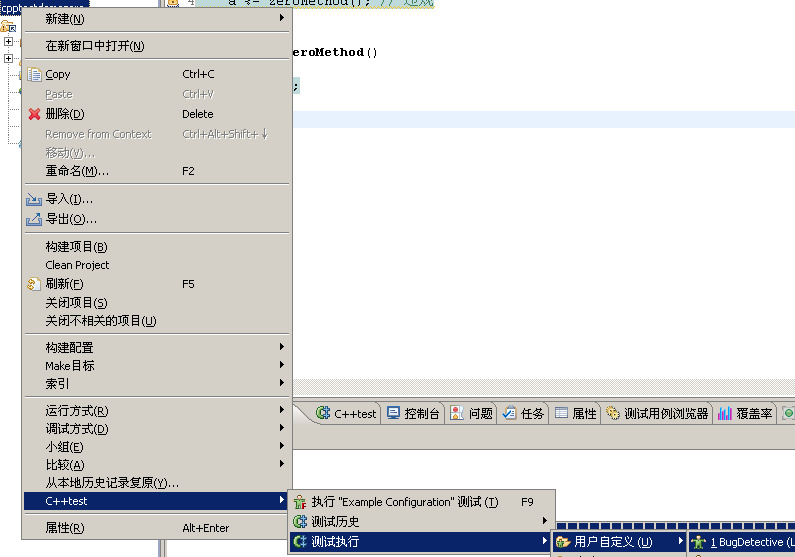


1. 点击复制之后，到“用户自定义”下的“BugDetective（License Required）” ，选择“静态” ，浏览规则，我们这里还是用最经典的测试条款来做demo：



然后“应用 → 关闭” 。

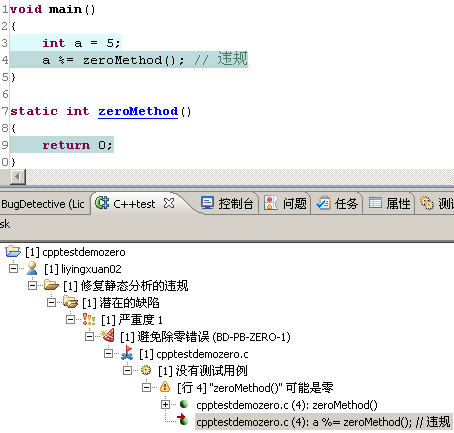
1. 执行测试：右键点击待测项目，选择“C++test → 测试执行 → 用户自定义” ，点击“1 BugDetective”即可：



1. 为了检测出“避免除零错误”这个测试项，我用了一段专用代码：

|  |
| --- |
| **void** **main**()  {  **int** a = 5;  a %= zeroMethod(); // 违规  }  **static** **int** **zeroMethod**()  {  **return** 0;  } |

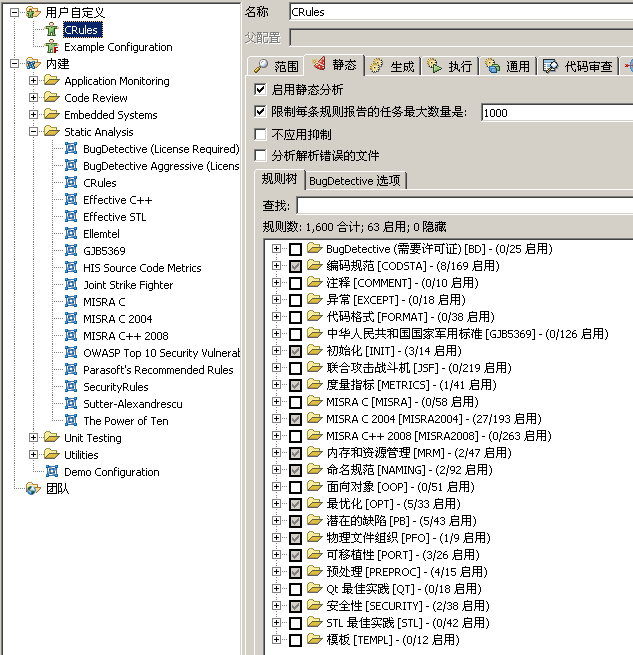
1. 然后看测试执行结果：



从上图，我们可以看见此BUG已经成功检验出。

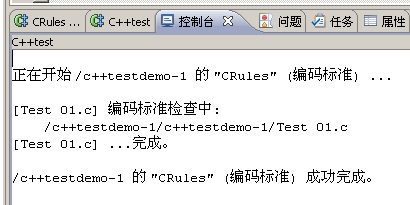
**二、静态分析 —— CRules**

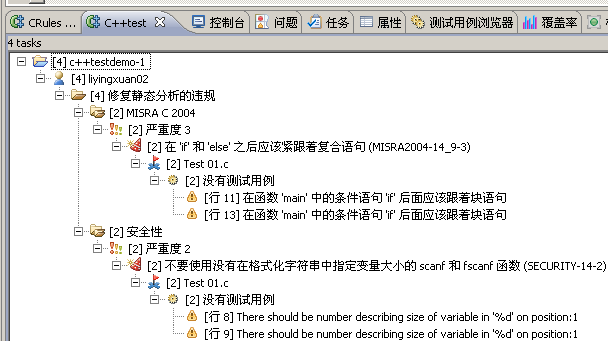
1. 点击“C++test → 测试配置” ，展开“内建 → Static Analysis” ，然后Right click on“CRules” ，选择“复制” ，点击复制之后，到“用户自定义”下的“CRules” ，选择“静态” ，浏览规则并修改：



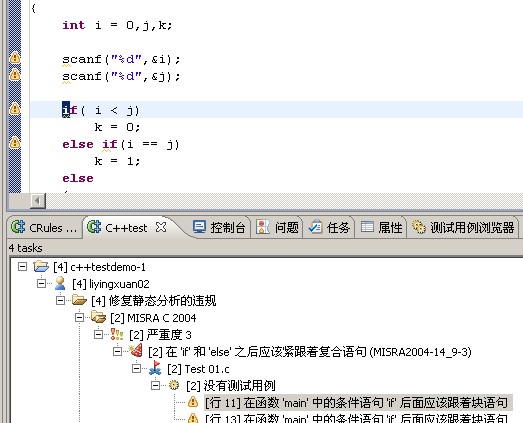
然后“应用 → 关闭” 。

1. 执行测试：右键点击待测项目，选择“C++test → 测试执行 → 用户自定义” ，点击“1 CRules”之后，会自动执行测试并生成测试结果，可以在“控制台”和“C++test”里看见：

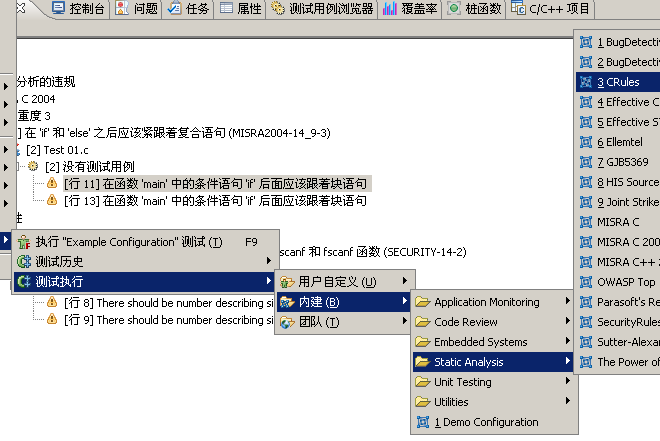




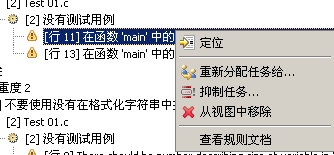
双击三角形黄色叹号后的行，可以直接链接到相应的代码处：



1. 如果不需要修改“内建”的规则，直接运行默认的测试配置，可以跳过复制那一步，直接运行测试，右键项目并选择“C++test → 测试执行 → 内建 → Static Analysis → CRules” ：

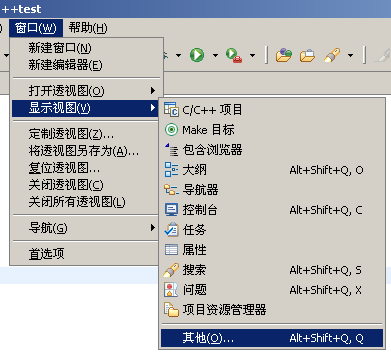


1. 右键点击问题列表中的具体错误信息：



我们可以通过选择“定位”链接到源码问题处，可以选“抑制任务”填入原因来绕过这条规则的测试，可以选择“查看规则文档”来阅读此条规则的详细内容等。

抑制任务列表：





点击确定后，可以看到软件下方的工作表区找到抑制列表。

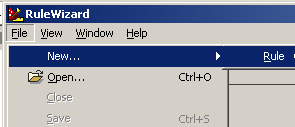
**三、自定义静态分析测试规则**

1. 定义规则：

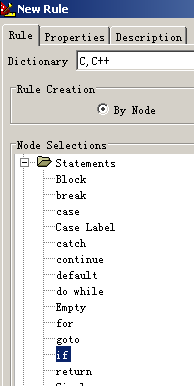
Click " C++test → 启动RuleWizard" :



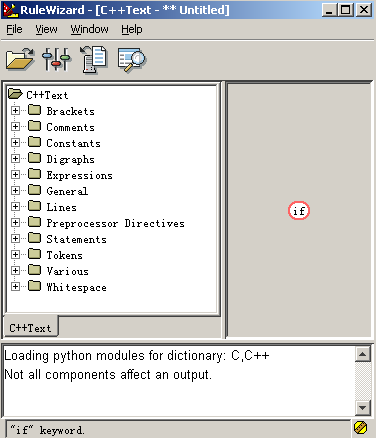
Click "File → New → Rule" :



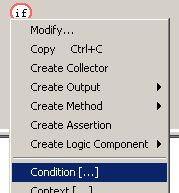
Select the "C,C++" , Click "By Node" , open up " Statements" , click "if" , click "OK" :



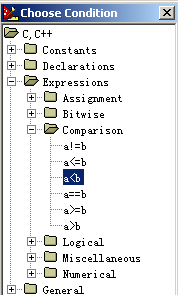
Then :



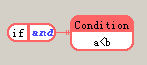
Right click on " if " , click " Condition[...] " :



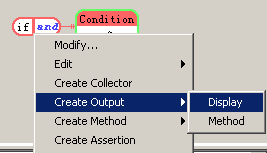
Then , Click " Expressions → Assignment → a<b → OK" :



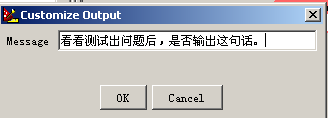
Then :



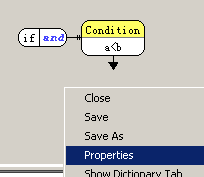
Right click on " if " , click " Create Output → Display " :



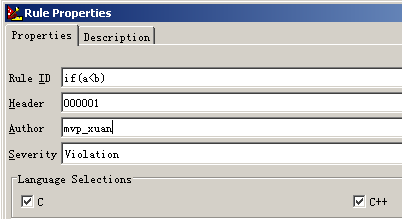
在Message中输入信息，click "OK" :



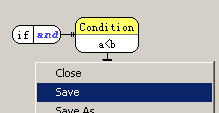
Right-click the empty space , select " Properties " :



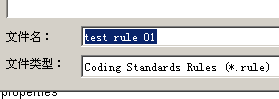
Input "Rule ID" , "Header" and "Author" ,click "OK" :



Right-click the empty space , select " Save " :



输入文件名，click "OK" :

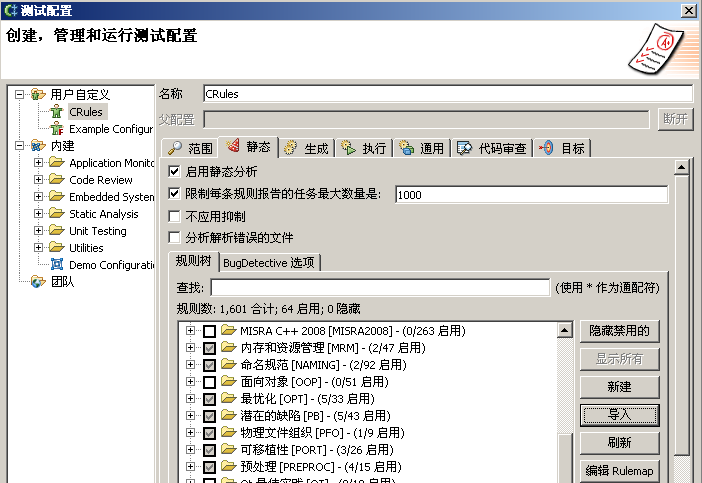


Then :

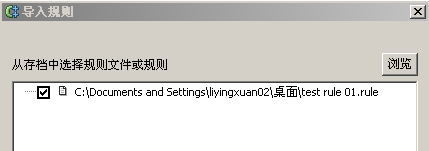


1. 导入自定义规则

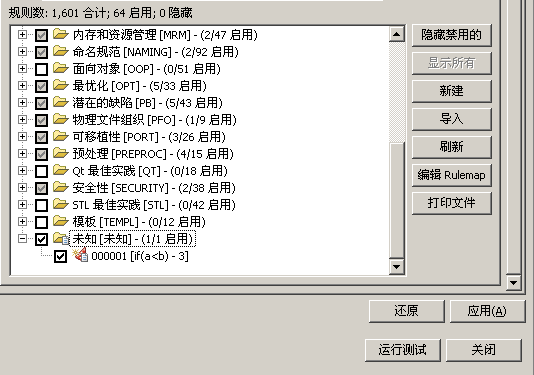
点击“C++test → 测试配置” ，选择用户自定义下的“CRules” ，选择“静态” ，点击“导入” ：



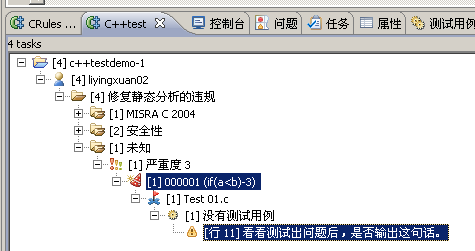
通过“浏览”找到刚才保存的rules文件，并点击“确定” ：



在规则列表中，找到新导入的，并选中它，点击“应用 → 关闭” ：



然后返回去再对待测源码执行一次CRules ：



刚才定义的规则和输入语言都出来了。

PS：这里我选择if和a<b来进行说明这个规则添加的原因是我的demo源码中有“ **if**( i < j)” ，所以定义这个规则，使这项测试有结果。

**四、 自动化白盒测试**(*程序员用来自己检测代码*)

**A. 自动建立测试驱动和桩函数**

C++Test自动建立一个测试驱动程序，其设计目标是极大化类的测试覆盖性和错误检测。为类建立测试驱动，你只要简单地打开这个类，然后按Build Test键。C++Test将自动建立测试驱动程序。   
另外，如果被测的方法需要调用当时还不存在或无法访问的函数，C++Test能够自动生成桩函数；这样能够测试与外部资源操作的交互作用和不包含任何隐藏的弱点。C++Test不是实际调用这些函数，而是调用桩函数并返回桩函数提供的值。如果你需要控制使用的返回值，你可以建立一个桩调用表，生命输入/输出的关系。

你还能加入用户定义的桩函数。例如，如果你要使用原始的函数，且该函数定义在不同的文件中；或者你想要仿真原始函数的行为，而用一个简单的函数替代它

自动生成C/C++类的测试驱动程序和桩函数的能力是C++Test所独有的；只有C++Test能够自动测试C/C++类（一当它能够编译时），而不需要用户的任何干预。使得你能够尽快地自动检测代码错误，以最容易、最省钱和最快速的方法找到和修正它们。如果没有这样的自动化工具，大量的时间和资源消耗将失去单元测试的潜在好处和现实意义。

**B. 白盒测试**

C++Test提供了一种有效并且高效的方法执行白盒测试。C++Test完全自动执行所有的白盒测试过程，自动生成和执行精心设计的测试用例。自动标记任何运行失败，并以一种简单的图示化结构显示。然后自动保存这些测试用例，能够方便地用于以后的回归测试。   
由于C++Test能够自动生成桩函数，或允许你加入自己的桩函数，因此它能够测试引用外部对象的类。换句话说，C++Test能够运行任何一个或一组类，并自动生成和执行一组测试用例，它们被设计成能够发现尽可能多的错误。

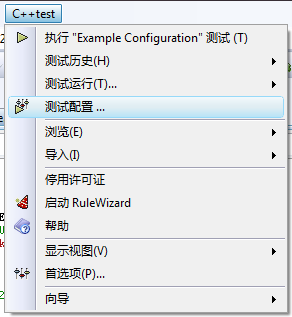
C++Test允许你定制白盒测试用例的生成，和在什么层次上（项目、文件、类或方法）执行测试。

示例一  
　 假设我们要测试如下一个类的成员函数：int mode2(int nParam)，则在进行白盒测试时软件会自动为我们生成如下6个测试用例：  
　　nParam = 1, 0, -1, 2147483647, -2147483647, 230  
  
　　可以看出，软件测试用例的生成主要还是测试一些边界值，例如最大值、最小值、0等。  
　　示例二  
　　假设我们要测试如下一个类的成员函数：void strcpy(char\* dest, char const \* src)，则软件会自动生成如下9个测试用例：  
　　(1) dest = NULL, src = NULL  
　　(2) dest = "yPqKIJ!u\_", src = NULL  
　　(3) dest = "", src = NULL  
　　(4) dest = NULL, src = "h)zn9b"  
　　(5) dest = "BsmC,/i=zI6CT}pX", src = "HcI{BeP(J"  
　　(6) dest = "", src = "% i?~TnON"  
　　(7) dest = NULL, src = ""  
　　(8) dest = "($MN<n;^", src = ""  
　　(9) dest = "", src = ""  
　　可见，如果我们的代码在实现时没有对各种可能情况（尤其是边界条件）进行特殊处理的话，则通过C++Test可以方便地发现这些潜在的问题。此外，对于一些特殊的测试情况，我们还可以手工创建测试用例。此外，采用C++ Test也可以帮助我们检查程序的编码情况，判断是否严格按编码规范进行开发。

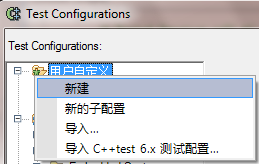
**C. 一个简单测试教程**

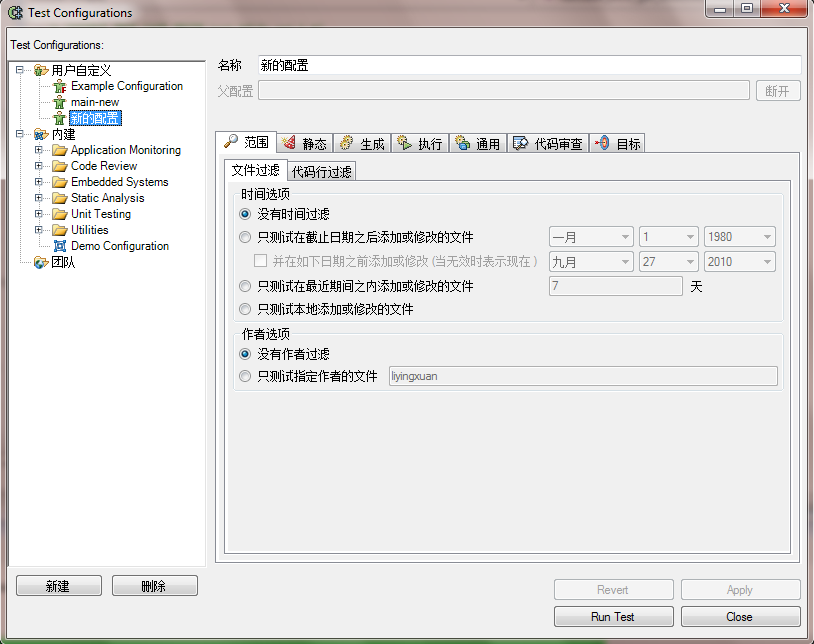
Anyway，我们来一次快速的自动生成Test case，并执行Test的流程：

1. Click“C++test → 测试配置” ：



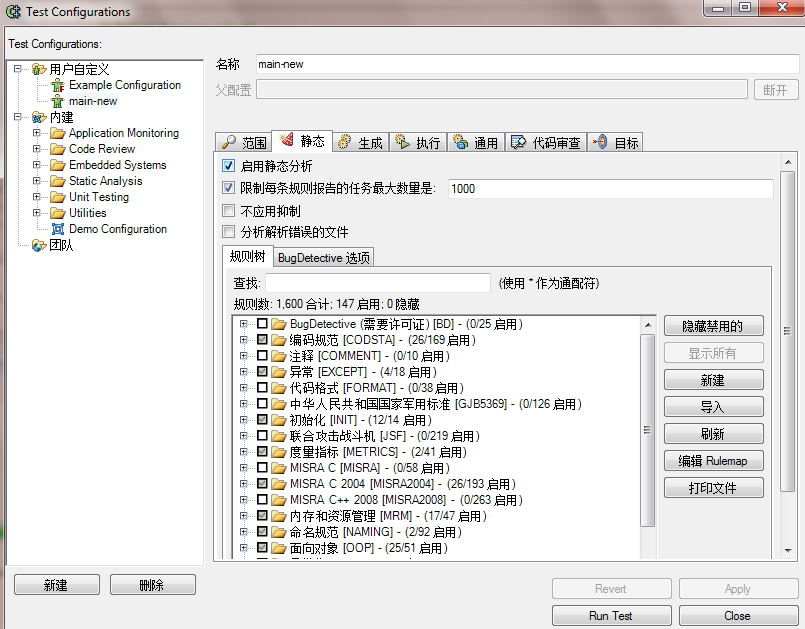
2. 然后进入了“Test Configurations” ，Right click on“用户自定义” ，Select the“新建” ，修改名称：



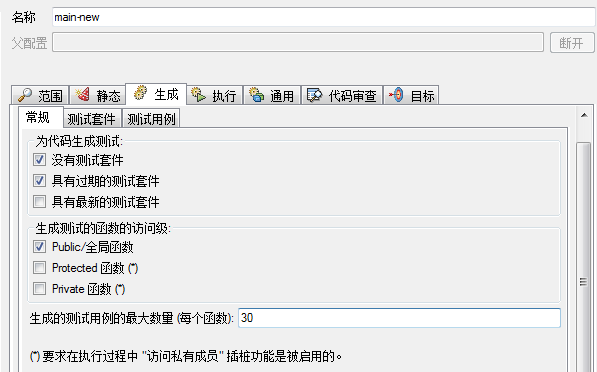


1. 然后可以看一下“Test Configurations”的全图，其实看到“内建” ，大家就应该知道了，其实C++test已经默认创建了各种测试类型，可以一项一项分门别类的测试，这就是developer自我测试时使用的了，也可以通过用户自定义，定义一些自己想要测试的项，而身为测试人员，是肯定需要自己手动创建一个个Test Solutions，闲话先不说，我们继续简单熟悉C++test。

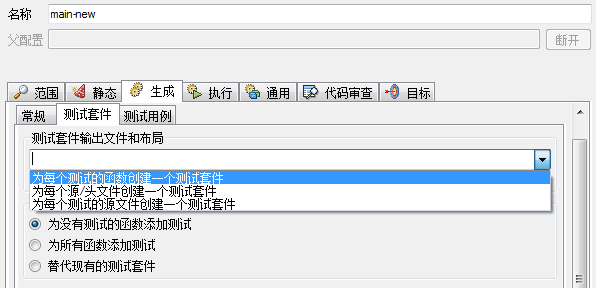
Select the“静态”项：



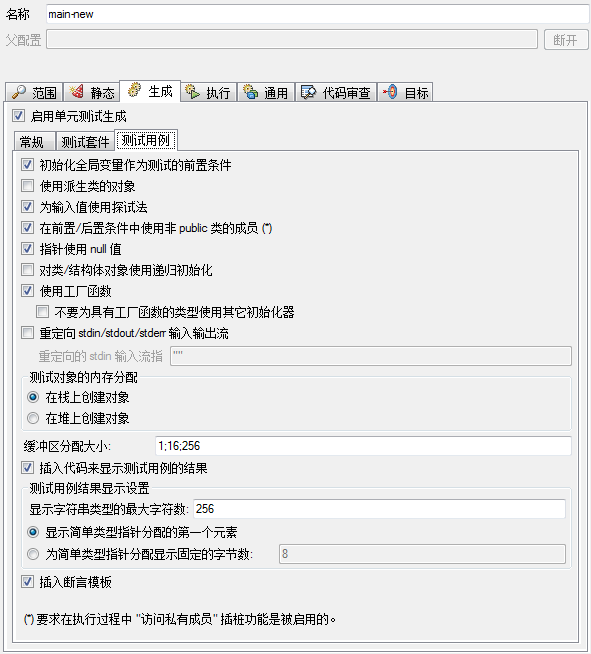
4. Select the“生成” ，按照下图所示选上相应选项：



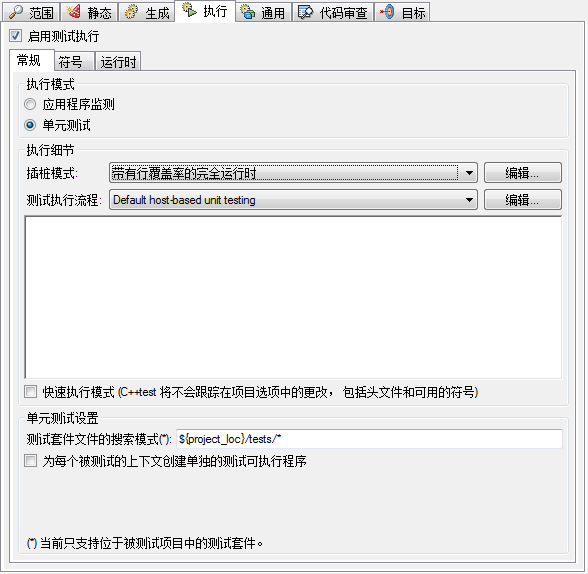
5. Select the“生成 → 测试套件” ，按照下图所示选上相应选项：



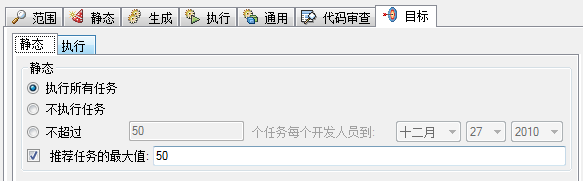
6. Select the“生成 → 测试用例” ，按照下图所示选上相应选项：



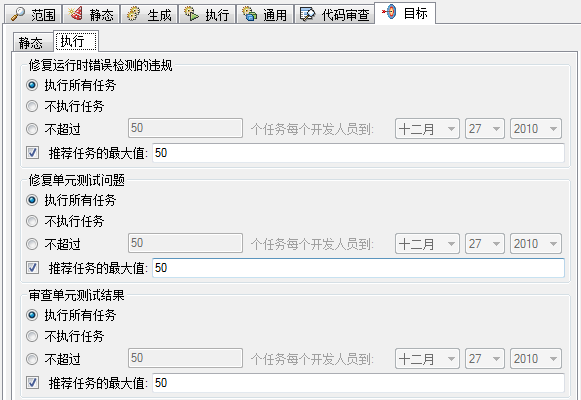
7. Select the“执行” ，按照下图所示选上相应选项：



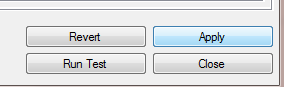
8. Select the“目标” ，按照下图所示选上相应选项：



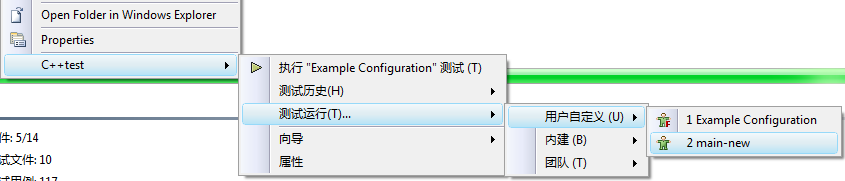
9. Select the“目标 → 执行” ，按照下图所示选上相应选项：



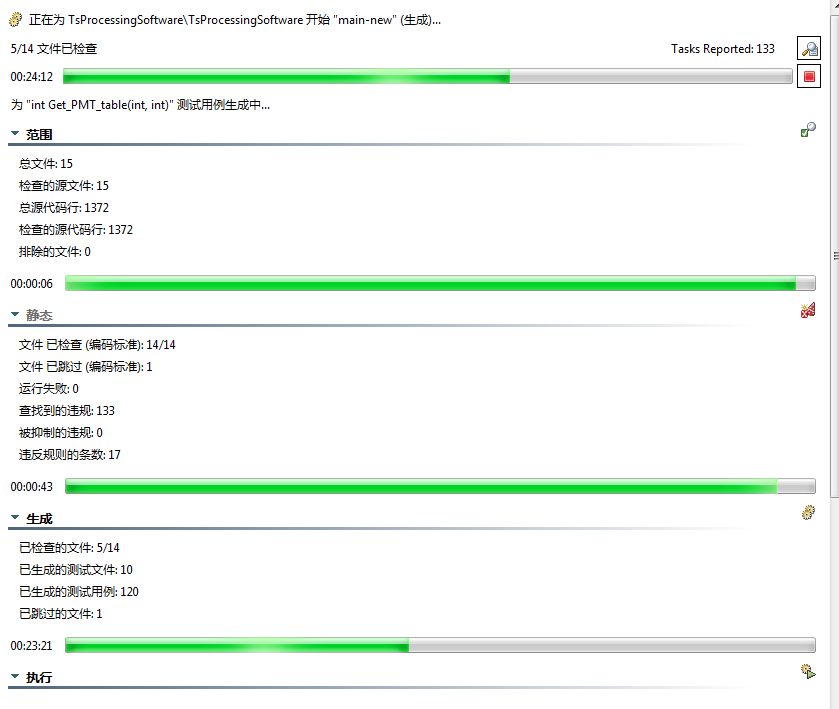
10. Click“Apply → Close” ：



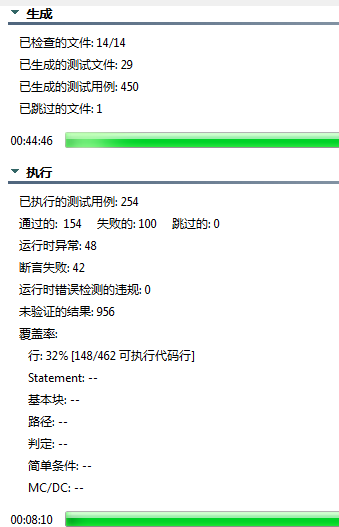
11. 在Solution Explorer下的项目上Right click，Select the“C++test → 测试运行 → 用户自定义 → 2 main-new（自定义的名称） ” ：



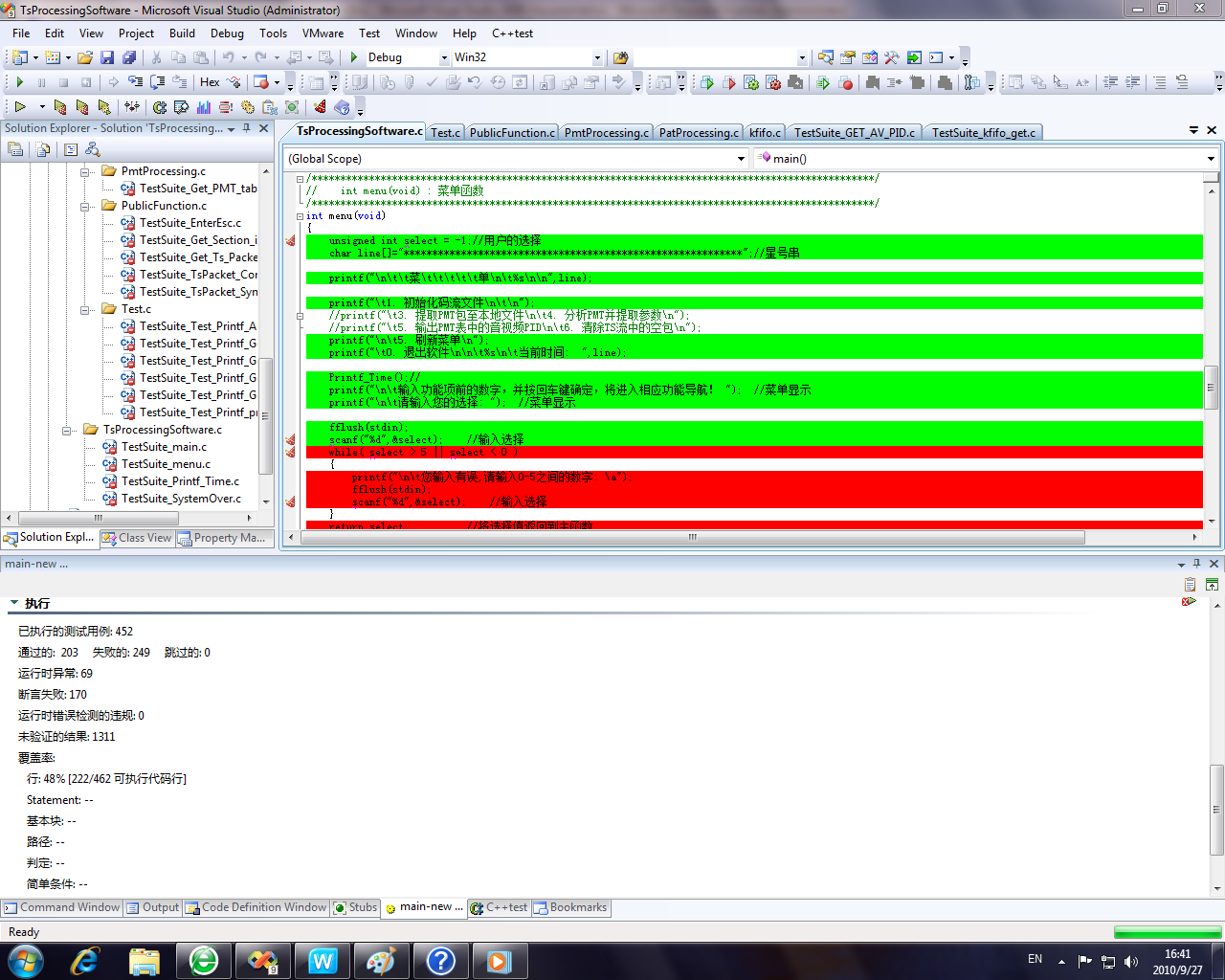
12. 测试开始执行了：



1. 到这里，你就使用C++test执行完一次完整的自动化静态分析和Unit Tests了。



1. 上图还没有执行完，后部没有截出来而已，下图是执行之后的：



1. 从上图可以看见，执行完毕后会告诉你行覆盖率，成功和失败的数量等信息，在源代码中会使用红色和绿色标注测试的判断结果。当然绿色的也未必一定对，红色也有可能是断言插入失败所致。

这里只是用了一个小小的item来做了一些test。后面几天会研究手动设计test case的一些flow。

PS：这里其实安装的C++test7.3VS2008Extension，并不是完整版，明天我会上传完整版的测试。今天没有带笔记本过来。

暂时先用变态的English OS 和Chinese C++test做研究，之后会用全English的做一个完整的Demo。