### CrashExporter是什么？

CrashExporter是基于开源项目[crashrpt](https://code.google.com/p/crashrpt/) 基础上改造的、基于Windows平台的轻量级异常导出组件。

原[crashrpt](https://code.google.com/p/crashrpt/) 主要将远端的崩溃信息压缩包通过网络发送到本地，而往往我们希望崩溃信息只在本地生成。同时[crashrpt](https://code.google.com/p/crashrpt/) 没有生成我们感兴趣的堆栈打印文件。

CrashExporter与crashrpt的区别是：CrashExporter只保留了crashrpt的导出dmp和抓屏功能，并增加了堆栈打印功能。

### 程序崩溃后会生成什么？

当程序崩溃后，会在程序执行的crashrpt目录下，增加一个以崩溃时间命名的文件夹。

文件夹里有三个文件：

crashdump.dmp为minidump文件；可以使用windbg工具查看（需要pdb文件一起使用，常用命令.ecxr和!analyze -v），或者直接用vs2010打开。

crashinfo.txt为堆栈打印；记录了异常时系统信息和堆栈信息打印，可以定位到代码文件的某一行。

screenshot0.png为屏幕截屏文件；抓取崩溃时的屏幕图像。

### 使用CrashExporter时，工程怎么配置？

主要是release工程的配置。

**对于VC6工程：**

1. Properties->C/C++->General中的Debug info，选择Program Database；
2. 设置Properties->C/C++->Code Generation中的Use run-time library为Multithreaded DLL;
3. Properties->C/C++->link中勾选上Generate debug info。

**对于vs2010工程：**

1. 使用多线程DLL(/MD)

设置方法，右击工程，选择属性->C/C++->代码生成，在当前页的“运行库”中选择“多线程 DLL(/MD)”。

1. 设置debugging symbols (PDB文件)

设置方法，右击工程，选择属性->C/C++->常规，在当前页“调试信息格式”中选择“程序数据库(/Zi)”

1. 打开链接器调试“生成调试信息”功能

设置方法，右击工程，选择属性->链接器->调试，在当前页“生成调试信息”中选择“是(/DEBUG)”。

### MFC 程序中如何使用CashExporter？

你可以重写CWinApp::Run()，在Run函数里安装CrashExporter。代码类似如下：

int CYourApp::Run()

{

// Call your crInstall code here ...

BOOL bRun;

BOOL bExit=FALSE;

while(!bExit)

{

bRun= CWinApp::Run();

bExit=TRUE;

}

return bRun;

}

### 如何在多线程程序中使用？

一般直接调用CrInstall即可获取所有线程异常信息。你也可以在每个线程的起始调用crInstallToCurrentThread来分别获取线程的异常信息，但是别忘记在线程退出是调用crUninstallFromCurrentThread。

### 程序僵死了，这种情况怎么处理？

这时候可以通过crGenerateErrorReport接口手动获取异常信息。

### CrashExporter支持程序自动重启么？

支持。 程序重启需要两个条件：

1）程序运行需要超过1分钟。

2）需要设置了CR\_INSTALL\_INFO中dwFlags|CR\_INST\_APP\_RESTART。

### 当程序发布时，我需要打包进哪些东西？

你至少需要打包进三个东西：

• CrashRpt.dll

• CrashExporter.exe

• dbghelp.dll

另外，你也必须确保客户机器安装了vs2010运行库。没有的话，可以让客户安装vs2010运行库，或者打包msvcp100.dll和msvcr100.dll。

### 我的程序是一个DLL，这种情况下怎么使用CrashExporter？

你可以在DLL程序初始化之前加载CrashExporter，比如DllMain()。 但是，调用DLL的主程序也可能加载了CrashExporter，这样情况下会屏蔽掉主线程的崩溃信息获取。这种冲突，就需要人为控制了，建议是在主程序中加载CrashExporter。

### 使用CrashExporter会不会影响我程序的性能？

不会！因为CrashExporter不会在后台执行任何任务，也不会占用多余的内存。CrashExporter仅仅会在程序崩溃的时候被调用。但是，CrashExporter会延长异常程序退出时间，因为CrashExporter需要加载symbols，分析堆栈，写dmp和调用堆栈文件。

### CrashExporter可以获取所有的异常吗？

可以获取大多数常见的异常。 因为安全原因，微软是不允许我们拦截某些异常的。 还有一种情况：

CPoint\* pt = new CPoint;

delete pt;

delete pt;

这种是因为堆损坏引起的，这种情况有时候会导致程序立即崩溃，有时候却是可以继续工作。对于这种随机的异常，CrashExporter是不能获取的。 因此，我们建议删除指针内存时，采用这样类似宏。

#define SAFE\_DELETE(ptr) { delete(ptr); (ptr)=NULL; }

### 我仔细检查了一切配置都是正确的，但是仍然不能获取到异常信息，这是为什么？

一个可能的原因是你异常的程序连给CrashExporter分配资源的机会都没给，比如你程序中有一个循环递归。

### 有没有异常信息指向CrashExporter本身出错的情况？比如跟踪异常堆栈跟踪到Crashrpt.dll。

有的。 当异常产生时，Crashrpt.dll需要检查异常指针的结构，这个指针结构一般由操作系统分配。但是，有时候这个指针结构会不存在（比如无效的参数），这时候Crashrpt.dll会根据当前CPU寄存器状态分配指针结构。这种情况下，你会在异常堆栈中跟踪到Crashrpt.dll了。

### 简述CrashExporter原理？

CrashExporter由两个core组成。分别为CrashRpt.dll 和 CrashExporter.exe。

CrashRpt.dll负责获取程序的异常信息并保存至共享内存；CrashExporter.exe负责从共享内存中获取异常信息，写dmp和堆栈调用文件、生成屏幕截屏。 这样做的目的是把容易出错的写文件等操作分离到CrashExporter.exe中，主程序只会加载CrashRpt.dll到其地址空间，这样减少了主程序因为CrashExporter出现异常的可能性。

CrashRpt.dll 和 CrashExporter.exe使用共享内存传递数据。

### 应该用什么类型的minidump？

建议用MiniDumpNormal，因为MiniDumpNormal包含了我们感兴趣的每个线程堆栈信息。

### 什么是异常？

异常（或者说关键性错误、崩溃）一般来说是你的程序运行不正常，从而不得不停止运行的情况。比如说，如果你的程序访问一块无效的内存地址（如NULL指针）、无法分配一个Buffer（内存不足）、C语言库的运行时（C run-time libraries，CRT）发现一个错误，并且需要程序立即停止运行等等，这些情况下都会产生一个异常。

### 哪些情况都会导致异常？

• 程序访问了一块非法的内存地址（比如NULL指针）.

•在无限递归中，栈溢出.

•大块数据被写入一片小缓冲区

•C++类中的纯虚函数被调用

•内存无法分配（内存不足）

•向C++的系统函数中传入非法的参数

•C运行库遇到错误，需要停止程序运行

主要有两种类型的异常：SEH异常（结构化异常处理）和标准C++异常

### 简述捕获SEH异常和标准C++异常。

结构化异常处理系统是由操作系统提供的（这意味着所有的Windows程序都能产生和处理SEH异常）。SEH异常最初是为C语言设计的，但在C++中也可以使用。

SEH异常是通过 try{} except(){} 这样的结构来处理的。程序中的 main() 函数就被这样的结构包围着，所以所有没有被处理的SEH异常默认都会被捕获，华生医生会弹出来。 SEH异常处理是由Visual C++编译器指定的。如果你要写兼容性强的代码，你应该在SEH结构两端加上#ifdef/#endif（就是说如果SEH没被定义，那么SEH的代码就不要参与编译）。

示例代码如下：

int\* p = NULL; // pointer to NULL

\_\_try

{

// Guarded code

\*p = 13; // causes an access violation exception

}

\_\_except(EXCEPTION\_EXECUTE\_HANDLER) // Here is exception filter expression

{

// Here is exception handler

// Terminate program

ExitProcess(1);

}

C++形式的异常处理系统是由是由C运行时库提供的（这意味着只有C++程序可以产生和处理这种异常）。C++形式异常处理是通过try{} catch{}这样的结构来处理的。

示例的代码如下：

try

{

throw 20;

}

catch (int e)

{

cout << "An exception occurred. Exception Nr. " << e << endl;

}