

## CoderZh的技术博客

一个程序员的思考与总结(请移步至：<http://blog.coderzh.com/>)

[博客园](#)[首页](#)[联系](#)[订阅](#)[管理](#)

## 公告

DigitalOcean优惠码

这里的博客将不再更新，最新博客

请移步至：

我的独立博客：

<http://blog.coderzh.com/>



微信公众号：

hacker-thinking

昵称：CoderZh

园龄：11年1个月

粉丝：790

关注：10

+加关注

## 搜索

找找看

## 随笔分类

[Agile\(2\)](#)

[Android\(3\)](#)

[ASP.NET\(3\)](#)

随笔-234 文章-10 评论-2047

## 玩转Google开源C++单元测试框架Google Test系列(gtest)之五 - 死亡测试

## 一、前言

“死亡测试”名字比较恐怖，这里的“死亡”指的是程序的崩溃。通常在测试过程中，我们需要考虑各种各样的输入，有的输入可能直接导致程序崩溃，这时我们就需要检查程序是否按照预期的方式挂掉，这也就是所谓的“死亡测试”。gtest的死亡测试能做到在一个安全的环境下执行崩溃的测试案例，同时又对崩溃结果进行验证。

## 二、使用的宏

Fatal assertion	Nonfatal assertion	Verifies
<code>ASSERT_DEATH(statement, regex`);</code>	<code>EXPECT_DEATH(statement, regex`);</code>	<i>statement</i> crashes with the given error
<code>ASSERT_EXIT(statement, predicate, regex`);</code>	<code>EXPECT_EXIT(statement, predicate, regex`);</code>	<i>statement</i> exits with the given error and its exit code matches <i>predicate</i>

由于有些异常只在Debug下抛出，因此还提供了\*\_DEBUG\_DEATH，用来处理Debug和Release下的不同。

## 三、\*\_DEATH(statement, regex`)

1. statement是被测试的代码语句

2. regex是一个正则表达式，用来匹配异常时在stderr中输出的内容

C#(20)  
C/C++(24)  
Cocos2d-x(1)  
Emacs(2)  
Google App Engine(7)  
JAVA(3)  
Linux(1)  
Lua(2)  
Python(66)  
Ubuntu(9)  
VBS(4)  
安全性测试(9)  
测试生活感悟(7)  
程序人生(15)  
代码安全(3)  
单元测试(19)  
公告(13)  
每周总结(4)  
软件测试(30)  
设计模式  
性能测试(7)  
学习笔记(27)

## 随笔档案

2015年9月 (1)  
2015年8月 (2)  
2015年6月 (4)  
2015年5月 (2)  
2015年4月 (5)  
2015年3月 (1)

如下面的例子:

```
void Foo()
{
    int *pInt = 0;
    *pInt = 42 ;
}

TEST(FooDeathTest, Demo)
{
    EXPECT_DEATH(Foo(), "");
}
```

重要：编写死亡测试案例时，TEST的第一个参数，即testcase\_name，请使用DeathTest后缀。原因是gtest会优先运行死亡测试案例，应该是为线程安全考虑。

## 四、\*\_EXIT(statement, predicate, regex`)

1. statement是被测试的代码语句
2. predicate 在这里必须是一个委托，接收int型参数，并返回bool。只有当返回值为true时，死亡测试案例才算通过。gtest提供了一些常用的predicate：

```
testing::ExitedWithCode(exit_code)
```

如果程序正常退出并且退出码与exit\_code相同则返回 true

```
testing::KilledBySignal(signal_number) // Windows下不支持
```

2014年5月 (2)  
2014年4月 (2)  
2011年5月 (1)  
2011年3月 (1)  
2011年1月 (1)  
2010年12月 (3)  
2010年11月 (3)  
2010年10月 (2)  
2010年9月 (6)  
2010年8月 (2)  
2010年7月 (4)  
2010年6月 (3)  
2010年5月 (4)  
2010年4月 (9)  
2010年3月 (6)  
2010年2月 (3)  
2010年1月 (16)  
2009年12月 (6)  
2009年11月 (3)  
2009年10月 (4)  
2009年9月 (3)  
2009年8月 (2)  
2009年7月 (7)  
2009年6月 (2)  
2009年4月 (12)  
2009年3月 (5)  
2009年2月 (2)  
2009年1月 (3)  
2008年12月 (7)

如果程序被signal\_number信号kill的话就返回true

3. regex是一个正则表达式，用来匹配异常时在stderr中输出的内容

这里，要说明的是，\*\_DEATH其实是对\*\_EXIT进行的一次包装，\*\_DEATH的predicate判断进程是否以非0退出码退出或被一个信号杀死。

例子：

```
TEST(ExitDeathTest, Demo)
{
    EXPECT_EXIT(_exit(1), testing::ExitedWithCode(1), "");
}
```

## 五、\*\_DEBUG\_DEATH

先来看定义：



```
#ifdef NDEBUG

#define EXPECT_DEBUG_DEATH(statement, regex) \
    do { statement; } while (false)

#define ASSERT_DEBUG_DEATH(statement, regex) \
    do { statement; } while (false)

#else

#define EXPECT_DEBUG_DEATH(statement, regex) \
    EXPECT_DEATH(statement, regex)
```

2008年11月 (9)  
2008年9月 (8)  
2008年8月 (7)  
2008年7月 (8)  
2008年6月 (9)  
2008年5月 (33)  
2008年4月 (6)  
2008年2月 (1)  
2007年12月 (3)  
2007年11月 (3)  
2007年10月 (7)  
2007年9月 (1)

## 系列文章

Python天天美味系列  
攻击方式学习系列  
瘦客户端那些事  
玩转gtest系列

## 读书笔记

Python网络编程  
xUnit Test Patterns  
卓有成效的程序员

## 友情链接

## 积分与排名

积分 - 547334  
排名 - 200

## 最新评论

```
#define ASSERT_DEBUG_DEATH(statement, regex) \  
    ASSERT_DEATH(statement, regex)  
  
#endif // NDEBUG for EXPECT_DEBUG_DEATH
```




可以看到，在Debug版和Release版本下，\*\_DEBUG\_DEATH的定义不一样。因为很多异常只会在Debug版本下抛出，而在Release版本下不会抛出，所以针对Debug和Release分别做了不同的处理。看gtest里自带的例子就明白了：



```
int DieInDebugElse12(int* sideeffect) {  
    if (sideeffect) *sideeffect = 12;  
#ifndef NDEBUG  
    GTEST_LOG_(FATAL, "debug death inside DieInDebugElse12()");  
#endif // NDEBUG  
    return 12;  
}  
  
TEST(TestCase, TestDieOr12WorksInDgbAndOpt)  
{  
    int sideeffect = 0;  
    // Only asserts in dbg.  
    EXPECT_DEBUG_DEATH(DieInDebugElse12(&sideeffect), "death");  
  
#ifdef NDEBUG  
    // opt-mode has sideeffect visible.  
    EXPECT_EQ(12, sideeffect);  
#else  
    // dbg-mode no visible sideeffect.  
    EXPECT_EQ(0, sideeffect);  
}
```

1. Re:gtest参数化测试代码示例  
博客园的链接改了，是这个地址：  
--canbeing
2. Re:玩转Google开源C++单元测试框架Google Test系列(gtest)之一  
- 初识gtest  
@xiao\_1bai编译的目标就是生成lib  
文件，你已经成功了。现在可以在  
你的项目引用gtestd.lib...  
--cnbloghzc
3. Re:ViEmuVS2013-3.2.1 破解  
安装失败，提示：  
所需要的.NET Framework 没有  
--xiake007
4. Re:玩转Google开源C++单元测试框架Google Test系列(gtest)之七  
- 深入解析gtest  
谢谢，作者哥哥，这么多章，最精彩这章。领教了，谢谢  
--\$JackChen
5. Re:最常用的Emacs的基本操作  
如果Emacs入手都算有些难度。。。那VIM怎么办？  
--震灵
6. Re:PyQt4学习资料汇总  
大神，我下了你的财务管理系统，  
那个数据库文件提示打不开，怎么  
解决啊~~急用~~  
--lzgst

```
#endif  
}  

```

六、关于正则表达式

在POSIX系统 ( [Linux](#), [Cygwin](#), 和 [Mac](#) ) 中，gtest的死亡测试中使用的是POSIX风格的正则表达式，想了解POSIX风格表达式可参考：

1. [POSIX extended regular expression](#)
2. [Wikipedia entry](#).

在Windows系统中，gtest的死亡测试中使用的是gtest自己实现的简单的正则表达式语法。相比POSIX风格，gtest的简单正则表达式少了很多内容，比如 ("x|y"), ("(xy)"), ("[xy]") 和("x{5,7}")都不支持。

下面是简单正则表达式支持的一些内容：

	matches any literal character c
\\d	matches any decimal digit
\\D	matches any character that's not a decimal digit
\\f	matches \\f
\\n	matches \\n
\\r	matches \\r
\\s	matches any ASCII whitespace, including \\n
\\S	matches any character that's not a whitespace
\\t	matches \\t

7. Re:ViEmuVS2013-3.2.1 破解  
为啥我的注册表中没有whole  
tomato  
  
--蓝域小兵
8. Re:玩转Google开源C++单元测试框架Google Test系列(gtest)之三  
- 事件机制  
由于没有加TEST 宏，输出结果如下：  
[=====] Running 0 tests from 0 test cases.[=====] 0 tests from 0 test c.....  
  
--喜马拉雅
9. Re:从CEGUI源码看代码规范  
好一个singleton！  
  
--chaosink
10. Re:使用UI Automation库用于UI  
自动化测试  
mark  
  
--大恒爸爸

阅读排行榜

1. 玩转Google开源C++单元测试框架Google Test系列(gtest)(总)  
(221480)
2. 玩转Google开源C++单元测试框架Google Test系列(gtest)之一 - 初识gtest(146539)
3. 玩转Google开源C++单元测试框架Google Test系列(gtest)之二 - 断言(105010)

\\w	matches \\w
\\w	matches any letter, _, or decimal digit
\\W	matches any character that \\w doesn't match
\\c	matches any literal character c, which must be a punctuation
.	matches any single character except \\n
A?	matches 0 or 1 occurrences of A
A*	matches 0 or many occurrences of A
A+	matches 1 or many occurrences of A
^	matches the beginning of a string (not that of each line)
\$	matches the end of a string (not that of each line)
xy	matches x followed by y

gtest定义两个宏，用来表示当前系统支持哪套正则表达式风格：

1. POSIX风格：GTEST\_USES\_POSIX\_RE = 1
2. Simple风格：GTEST\_USES\_SIMPLE\_RE=1

七、死亡测试运行方式

1. fast方式（默认的方式）

```
testing::FLAGS_gtest_death_test_style = "fast";
```

2. threadsafe方式

4. 玩转Google开源C++单元测试框架Google Test系列(gtest)之三 - 事件机制(68517)
5. 玩转Google开源C++单元测试框架Google Test系列(gtest)之六 - 运行参数(54532)
6. 玩转Google开源C++单元测试框架Google Test系列(gtest)之四 - 参数化(54400)
7. C# 中使用JSON -DataContractJsonSerializer(52402)
8. PyQt4学习资料汇总(49575)
9. 玩转Google开源C++单元测试框架Google Test系列(gtest)之七 - 深入解析gtest(49311)
10. 代码覆盖率浅谈(47554)

## 评论排行榜

1. PyQt4学习资料汇总(152)
2. 开源Granados介绍 - SSH连接远程Linux服务器(C#)(66)
3. (原创)攻击方式学习之(1) - 跨站式脚本(Cross-Site Scripting) (49)
4. 三年之痒(44)
5. NancyBlog - 我的Google App Engine Blog(42)
6. 创业三年来的一些感想 - 游戏篇(40)
7. CCNET+MSBuild+SVN实时构建的优化总结(40)

```
testing::FLAGS_gtest_death_test_style = "threadsafe";
```

你可以在 main() 里为所有的死亡测试设置测试形式，也可以为某次测试单独设置。Google Test会在每次测试之前保存这个标记并在测试完成后恢复，所以你不需要去管这部分工作。如：



```
TEST(MyDeathTest, TestOne) {  
    testing::FLAGS_gtest_death_test_style = "threadsafe";  
    // This test is run in the "threadsafe" style:  
    ASSERT_DEATH(ThisShouldDie(), "");  
}  
  
TEST(MyDeathTest, TestTwo) {  
    // This test is run in the "fast" style:  
    ASSERT_DEATH(ThisShouldDie(), "");  
}
```

```
int main(int argc, char** argv) {  
    testing::InitGoogleTest(&argc, argv);  
    testing::FLAGS_gtest_death_test_style = "fast";  
    return RUN_ALL_TESTS();  
}
```



## 八、注意事项

1. 不要在死亡测试里释放内存。
2. 在父进程里再次释放内存。

8. CoderZh首款Python联机对战游戏 - NancyTetris1.0倾情发布 (一) (37)
9. 代码安全系列(1) - Log的注入(35)
10. 程序员的信仰(35)

## 推荐排行榜

1. 玩转Google开源C++单元测试框架Google Test系列(gtest)(总)(24)
2. 创业三年来的一些感想 - 游戏篇 (14)
3. 程序员的共鸣 - 读《卓有成效的程序员》(12)
4. 代码覆盖率浅谈(12)
5. 《xUnit Test Patterns》学习笔记 5 - xUnit基础(10)
6. 三年之痒(9)
7. 玩转Google开源C++单元测试框架Google Test系列(gtest)之一 - 初识gtest(9)
8. 优美的测试代码 - 行为驱动开发(BDD)(8)
9. Python天天美味(总)(7)
10. PyQt4学习资料汇总(6)

3. 不要在程序中使用内存堆检查。

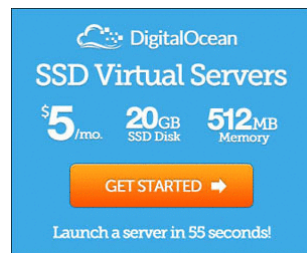
## 九、总结

关于死亡测试，gtest官方的文档已经很详细了，同时在源码中也有大量的示例。如想了解更多的请参考官方的文档，或是直接看gtest源码。

简单来说，通过\*\_DEATH(statement, regex)和\*\_EXIT(statement, predicate, regex)，我们可以非常方便的编写导致崩溃的测试案例，并且在不影响其他案例执行的情况下，对崩溃案例的结果进行检查。

系列链接：

1. 玩转Google开源C++单元测试框架Google Test系列(gtest)之一 - 初识gtest
2. 玩转Google开源C++单元测试框架Google Test系列(gtest)之二 - 断言
3. 玩转Google开源C++单元测试框架Google Test系列(gtest)之三 - 事件机制
4. 玩转Google开源C++单元测试框架Google Test系列(gtest)之四 - 参数化
5. 玩转Google开源C++单元测试框架Google Test系列(gtest)之五 - 死亡测试
6. 玩转Google开源C++单元测试框架Google Test系列(gtest)之六 - 运行参数
7. 玩转Google开源C++单元测试框架Google Test系列(gtest)之七 - 深入解析gtest
8. 玩转Google开源C++单元测试框架Google Test系列(gtest)之八 - 打造自己的单元测试框架



DigitalOcean的VPS主机，稳定、速度快、价格也实惠。可以在上面部署独立网站或各种实用工具。

我用了很久了，确实不错，极力推荐。

使用这个链接购买可获得10美元优惠。

优惠链接：[DigitalOcean优惠码](#)





微信扫一扫交流

作者：[CoderZh](#)

公众号：hacker-thinking（一个程序员的思考）

独立博客：<http://blog.coderzh.com>

博客园博客将不再更新，请关注我的「微信公众号」或「独立博客」。

作为一个程序员，思考程序的每一行代码，思考生活的每一个细节，思考人生的每一种可能。

文章版权归本人所有，欢迎转载，但未经作者同意必须保留此段声明，且在文章页面明显位置给出原文连接，否则保留追究法律责任的权利。

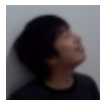
分类: [单元测试](#), [C/C++](#)

标签: [Google Test](#)

好文要顶

关注我

收藏该文



CoderZh

关注 - 10

粉丝 - 790

+加关注

0

0

« 上一篇: [Google App Engine已经支持JAVA了](#)

» 下一篇: [玩转Google开源C++单元测试框架Google Test系列\(gtest\)之六 - 运行参数](#)

posted @ 2009-04-08 23:56 CoderZh 阅读(40388) 评论(5) 编辑 收藏

## 评论列表

#1楼 2009-10-11 12:04 lizzoe

我已经包含了#include "gtest/gtest-death-test.h", 可是为什么还是编译不过呢? 谢谢

正在编译...

gtest\_demo.cpp

c:\documents and settings\administrator\桌面\gtest-1.3.0\gtest\_demo\gtest\_demo.cpp(24) : error C3861:

"EXPECT\_DEATH": 即使使用参数相关的查找,也未找到标识符

---

#### #2楼[楼主] 2009-10-18 13:45 CoderZh

@ lizzoe

你#include "gtest/gtest.h"就好了

支持(0) 反对(0)

#### #3楼 2009-10-26 23:46 我爱一条蚕

@ CoderZh

谢谢楼主,呵呵,我的问题在于,我用的是VC2003.NET,不是2005或者更高的,所以失败了。囧.....

支持(0) 反对(0)

---

#### #4楼[楼主] 2009-10-27 22:15 CoderZh

@ 我爱一条蚕

客气了,呵呵

支持(0) 反对(0)

---

#### #5楼 2013-01-06 18:45 水石心鱼

楼主你好:

```
void FooCopy(int n)
{
    char p[5];
    strcpy_s(p,n,"abc");
}
```

```
TEST(FooCopyDeathTest, DeathDemo)
{
    EXPECT_DEATH(FooCopy(1,"");
}
```

在启动调试后

会跳出Express ( Lbuffer is too small && 0 ) 的警告框

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】腾讯云上实验室 1小时搭建人工智能应用

【推荐】可嵌入您系统的“在线Excel”！SpreadJS 纯前端表格控件

【推荐】阿里云“全民云计算”优惠升级



#### 最新IT新闻:

- iOS 11正式版发布：可以升级了！
  - 国行iPhone 8/8 Plus首发开箱！双玻璃果然漂亮
  - Windows 10新版16291发布：小娜支持多设备断点续读
  - 卡西尼号：不是永别，是远方的游子回家
  - 乐视网：贾跃亭未按承诺将减持资金借予公司使用
- » 更多新闻...



#### 最新知识库文章:

- Google 及其云智慧
- 做到这一点，你也可以成为优秀的程序员
- 写给立志做码农的大学生

- [架构腐化之谜](#)
- [学会思考，而不只是编程](#)
- » [更多知识库文章...](#)

站长统计