C++单元测试、一种简单打桩方式的实现

此篇文章涉及大量代码示例更适合作为参考手册，需要时查查用法。对于技术人员代码是最好的解释。

**桩**，或称桩代码，是指用来代替关联代码或者未实现代码的代码。如果用函数B1来代替B，那么，B称为原函数，B1称为桩函数。打桩就是调用B的地方变成调用B1。

打桩主要涉及两点：

* 第一如何获取原函数地址
* 第二如何用桩函数替换原函数

## 桩函数替换原函数的原理介绍

主要用到inline hook技术，核心思想，通过替换目标函数头部指令，实现在函数执行之前跳转到其他的指令区域，执行完毕跳转回到原来的函数，跳转到的指令区域通常是我们自己编写的函数。

图1所示，如果原函数和桩函数同在32地址空间里，则采用JMP指令实现，占5字节。

图2所示，如果原函数和桩函数不同在32地址空间里，则采用MOV、PUSH、RET指令实现，占12字节。

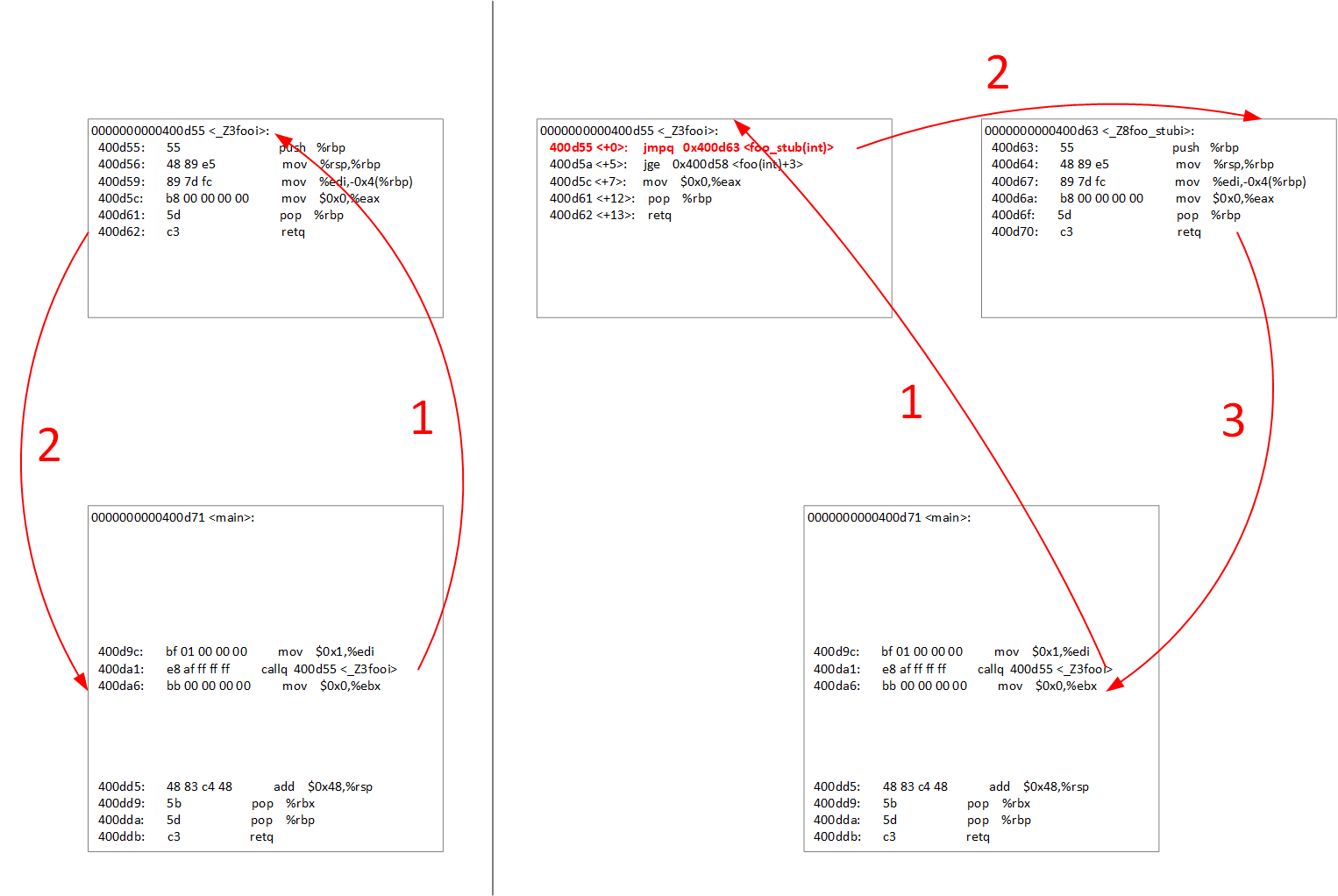


图1

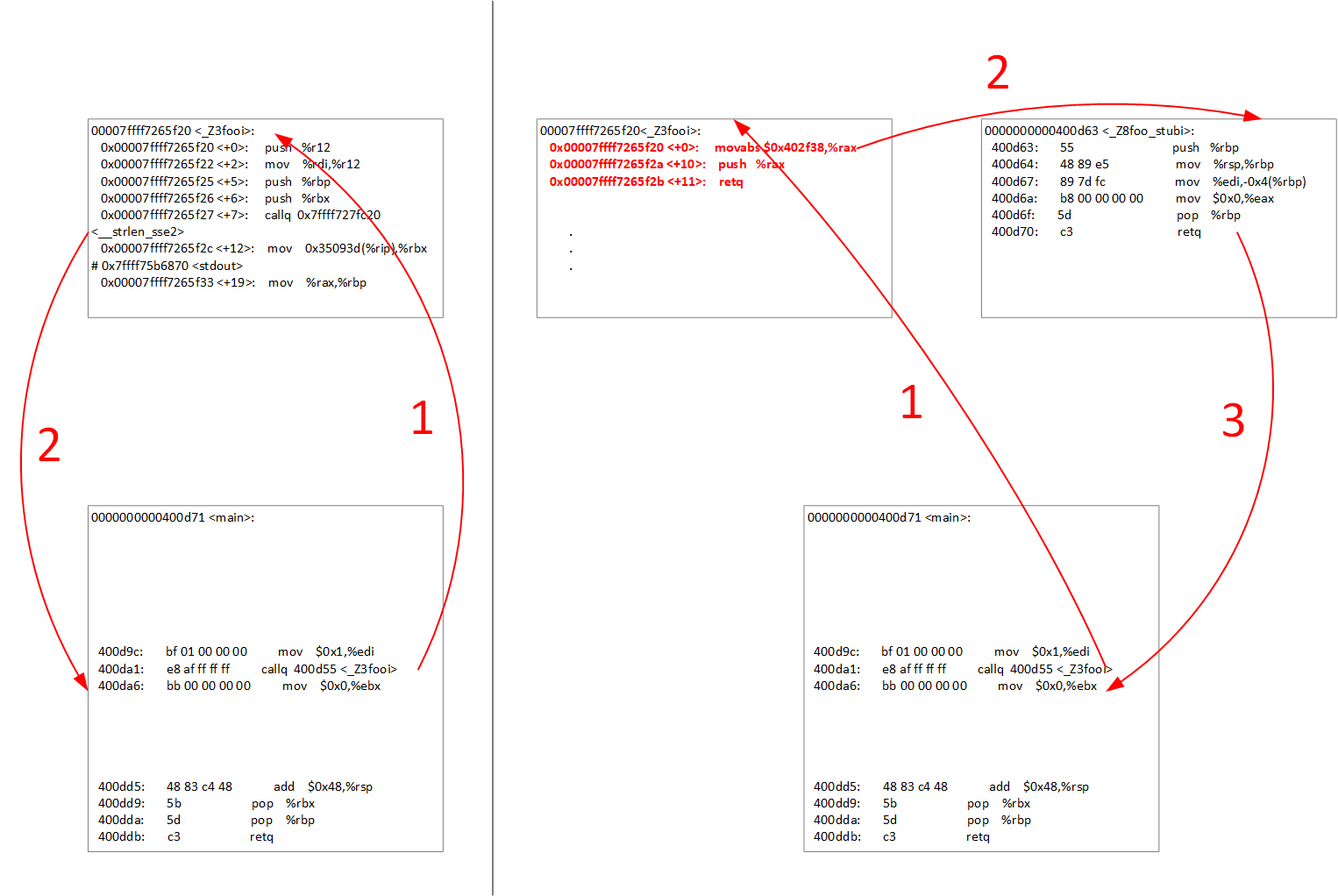


图2

**关键替换源码解释：**

template<typename T,typename S>

void set(T addr, S addr\_stub) **//addr原函数地址，addr\_stub桩函数地址**

{

void \* fn;

void \* fn\_stub;

fn = addrof(addr); **//强转地址**

fn\_stub = addrof(addr\_stub); **//强转地址**

struct func\_stub \*pstub;

pstub = new func\_stub;

//start

pstub->fn = fn;

#ifdef \_\_x86\_64\_\_

if(judge\_far\_jmp(fn, fn\_stub)) **//判断远跳还是近跳**

{

pstub->far\_jmp = true;

memcpy(pstub->code\_buf, fn, CODESIZE\_MAX);**//保留原函数现场**

}

else

{

pstub->far\_jmp = false;

memcpy(pstub->code\_buf, fn, CODESIZE\_MIN); **//保留原函数现场**

}

#else

memcpy(pstub->code\_buf, fn, CODESIZE);

#endif

#ifdef \_WIN32

DWORD lpflOldProtect;

if(0 == VirtualProtect(pageof(pstub->fn), m\_pagesize \* 2, PAGE\_EXECUTE\_READWRITE, &lpflOldProtect))

#else

if (-1 == mprotect(pageof(pstub->fn), m\_pagesize \* 2, PROT\_READ | PROT\_WRITE | PROT\_EXEC))

#endif

{

throw("stub set mprotect to w+r+x faild");

}

#ifdef \_\_x86\_64\_\_

if(pstub->far\_jmp)

{

**//12 byte替换，MOV、PUSH、RET**

**\*(unsigned char\*)fn = 0x48;**

**\*((unsigned char\*)fn + 1) = 0xb8;**

**\*(unsigned long long \*)((unsigned char \*)fn + 2) = (unsigned long long)fn\_stub;**

**\*(unsigned char \*)((unsigned char \*)fn + 10) = 0x50;**

**\*(unsigned char \*)((unsigned char \*)fn + 11) = 0xc3;**

}

else

{

**//5 byte替换，JMP指令**

**\*(unsigned char \*)fn = (unsigned char)0xE9;**

**\*(unsigned int \*)((unsigned char \*)fn + 1) = (unsigned char \*)fn\_stub - (unsigned char \*)fn - CODESIZE\_MIN;**

}

#else

**//5 byte替换，JMP指令**

**\*(unsigned char \*)fn = (unsigned char)0xE9;**

**\*(unsigned int \*)((unsigned char \*)fn + 1) = (unsigned char \*)fn\_stub - (unsigned char \*)fn - CODESIZE;**

#endif

#ifdef \_WIN32

if(0 == VirtualProtect(pageof(pstub->fn), m\_pagesize \* 2, PAGE\_EXECUTE\_READ, &lpflOldProtect))

#else

if (-1 == mprotect(pageof(pstub->fn), m\_pagesize \* 2, PROT\_READ | PROT\_EXEC))

#endif

{

throw("stub set mprotect to r+x failed");

}

m\_result.insert(std::pair<void\*,func\_stub\*>(fn,pstub));

return;

}

## 获取原函数地址，各种函数的打桩用法

各种类函数地址的获取方式各不相同，不同平台同种类的获取方式也不同，下面将会列举一些常见类型函数的地址获取方式。

桩函数写法基于调用约定，C++中常见的调用约定有stdcall、cdecl、fastcall和thiscall

* **普通函数打桩(非static)**

//for linux and windows

#include<iostream>

#include "stub.h"

using namespace std;

int foo(int a)

{

cout<<"I am foo"<<endl;

return 0;

}

int foo\_stub(int a)

{

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set(foo, foo\_stub);

foo(1);

return 0;

}

* **静态成员函数打桩**

//for linux and windows

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

static int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(int a)

{

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set(ADDR(A,foo), foo\_stub);

A::foo(1);

return 0;

}

* **实例成员函数打桩**

//for linux，\_\_cdecl

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(void\* obj, int a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set(ADDR(A,foo), foo\_stub);

A a;

a.foo(1);

return 0;

}

//for windows，\_\_thiscall

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

class B{

public:

int foo\_stub(int a){

cout<<"I am foo\_stub"<<endl;

return 0;

}

};

int main()

{

Stub stub;

stub.set(ADDR(A,foo), ADDR(B,foo\_stub));

A a;

a.foo(1);

return 0;

}

* **模板函数打桩(实例成员函数)**

//for linux，\_\_cdecl

#include<iostream>

#include "stub.h"

using namespace std;

class A{

public:

template<typename T>

int foo(T a)

{

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(void\* obj, int x)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

Stub stub;

stub.set((int(A::\*)(int))ADDR(A,foo), foo\_stub);

A a;

a.foo(5);

return 0;

}

//for windows，\_\_thiscall

#include<iostream>

#include "stub.h"

using namespace std;

class A{

public:

template<typename T>

int foo(T a)

{

cout<<"I am A\_foo"<<endl;

return 0;

}

};

class B {

public:

int foo\_stub(int a) {

cout << "I am foo\_stub" << endl;

return 0;

}

};

int main()

{

Stub stub;

stub.set((int(A::\*)(int))ADDR(A,foo), ADDR(B, foo\_stub));

A a;

a.foo(5);

return 0;

}

* **重载函数打桩(实例成员函数)**

//for linux，\_\_cdecl

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

int foo(int a){

cout<<"I am A\_foo\_int"<<endl;

return 0;

}

int foo(double a){

cout<<"I am A\_foo-double"<<endl;

return 0;

}

};

int foo\_stub\_int(void\* obj,int a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub\_int"<< a << endl;

return 0;

}

int foo\_stub\_double(void\* obj,double a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub\_double"<< a << endl;

return 0;

}

int main()

{

Stub stub;

stub.set((int(A::\*)(int))ADDR(A,foo), foo\_stub\_int);

stub.set((int(A::\*)(double))ADDR(A,foo), foo\_stub\_double);

A a;

a.foo(5);

a.foo(1.1);

return 0;

}

//for windows，\_\_thiscall

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int i;

public:

int foo(int a){

cout<<"I am A\_foo\_int"<<endl;

return 0;

}

int foo(double a){

cout<<"I am A\_foo-double"<<endl;

return 0;

}

};

class B{

int i;

public:

int foo\_stub\_int(int a)

{

cout << "I am foo\_stub\_int" << a << endl;

return 0;

}

int foo\_stub\_double(double a)

{

cout << "I am foo\_stub\_double" << a << endl;

return 0;

}

};

int main()

{

Stub stub;

stub.set((int(A::\*)(int))ADDR(A,foo), ADDR(B, foo\_stub\_int));

stub.set((int(A::\*)(double))ADDR(A,foo), ADDR(B, foo\_stub\_double));

A a;

a.foo(5);

a.foo(1.1);

return 0;

}

* **虚函数打桩**

//for linux

#include<iostream>

#include "stub.h"

using namespace std;

class A{

public:

virtual int foo(int a){

cout<<"I am A\_foo"<<endl;

return 0;

}

};

int foo\_stub(void\* obj,int a)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int main()

{

typedef int (\*fptr)(A\*,int);

fptr A\_foo = (fptr)(&A::foo); //获取虚函数地址

Stub stub;

stub.set(A\_foo, foo\_stub);

A a;

a.foo();

return 0;

}

//for windows x86(32位)

#include<iostream>

#include "stub.h"

using namespace std;

class A {

public:

virtual int foo(int a) {

cout << "I am A\_foo" << endl;

return 0;

}

};

class B {

public:

int foo\_stub(int a)

{

cout << "I am foo\_stub" << endl;

return 0;

}

};

int main()

{

unsigned long addr;

\_asm {mov eax, A::foo}

\_asm {mov addr, eax}

Stub stub;

stub.set(addr, ADDR(B, foo\_stub));

A a;

a.foo(1);

return 0;

}

//for windows x64(64位)，VS编译器不支持内嵌汇编。可以把汇编代码独立成一个文件。

* **内联函数打桩**

//for linux

//添加-fno-inline编译选项，禁止内联，能获取到函数地址，打桩参考上面。

//for windows

//添加/Ob0禁用内联展开。

* **第三方库私有成员函数打桩**

//for linux

//被测代码添加-fno-access-private编译选项，禁用访问权限控制，成员函数都为公有的

//无源码的动态库或静态库无法自己编译，需要特殊技巧获取函数地址

#include<iostream>

#include "stub.h"

#include "addr\_pri.h" //只适用c++11

using namespace std;

class A{

int a;

int foo(int x){

cout<<"I am A\_foo "<< a << endl;

return 0;

}

static int b;

static int bar(int x){

cout<<"I am A\_bar "<< b << endl;

return 0;

}

};

ACCESS\_PRIVATE\_FIELD(A, int, a);

ACCESS\_PRIVATE\_FUN(A, int(int), foo);

ACCESS\_PRIVATE\_STATIC\_FIELD(A, int, b);

ACCESS\_PRIVATE\_STATIC\_FUN(A, int(int), bar);

int foo\_stub(void\* obj, int x)

{

A\* o= (A\*)obj;

cout<<"I am foo\_stub"<<endl;

return 0;

}

int bar\_stub(int x)

{

cout<<"I am bar\_stub"<<endl;

return 0;

}

int main()

{

A a;

auto &A\_a = access\_private\_field::Aa(a);

auto &A\_b = access\_private\_static\_field::A::Ab();

A\_a = 1;

A\_b = 10;

call\_private\_fun::Afoo(a,1);

call\_private\_static\_fun::A::Abar(1);

auto A\_foo= get\_private\_fun::Afoo();

auto A\_bar = get\_private\_static\_fun::A::Abar();

Stub stub;

stub.set(A\_foo, foo\_stub);

stub.set(A\_bar, bar\_stub);

call\_private\_fun::Afoo(a,1);

call\_private\_static\_fun::A::Abar(1);

return 0;

}

//for windows，\_\_thiscall

#include<iostream>

#include "stub.h"

using namespace std;

class A{

int a;

int foo(int x){

cout<<"I am A\_foo "<< a << endl;

return 0;

}

static int b;

static int bar(int x){

cout<<"I am A\_bar "<< b << endl;

return 0;

}

};

ACCESS\_PRIVATE\_FIELD(A, int, a);

ACCESS\_PRIVATE\_FUN(A, int(int), foo);

ACCESS\_PRIVATE\_STATIC\_FIELD(A, int, b);

ACCESS\_PRIVATE\_STATIC\_FUN(A, int(int), bar);

class B {

public:

int foo\_stub(int x)

{

cout << "I am foo\_stub" << endl;

return 0;

}

};

int bar\_stub(int x)

{

cout<<"I am bar\_stub"<<endl;

return 0;

}

int main()

{

A a;

auto &A\_a = access\_private\_field::Aa(a);

auto &A\_b = access\_private\_static\_field::A::Ab();

A\_a = 1;

A\_b = 10;

call\_private\_fun::Afoo(a,1);

call\_private\_static\_fun::A::Abar(1);

auto A\_foo= get\_private\_fun::Afoo();

auto A\_bar = get\_private\_static\_fun::A::Abar();

Stub stub;

stub.set(A\_foo, ADDR(B,foo\_stub));

stub.set(A\_bar, bar\_stub);

call\_private\_fun::Afoo(a,1);

call\_private\_static\_fun::A::Abar(1);

return 0;

}

* **static函数打桩**

//for linux

#include <iostream>

#include <string>

#include <stdio.h>

#include "addr.h"

#include "stub.h"

// g++ -g test\_addr.cpp -std=c++11 -I../ -o test\_addr

static int test\_test()

{

printf("test\_test\n");

return 0;

}

static int xxx\_stub()

{

std::cout << "xxx\_stub" << std::endl;

return 0;

}

int main(int argc, char \*\*argv)

{

std::string res;

get\_exe\_pathname(res);

std::cout << res << std::endl;

unsigned long base\_addr;

get\_lib\_pathname\_and\_baseaddr("libc-2.17.so", res, base\_addr);

std::cout << res << base\_addr << std::endl;

std::map<std::string,ELFIO::Elf64\_Addr> result;

get\_weak\_func\_addr(res, "^puts$", result);

test\_test();

Stub stub;

std::map<std::string,ELFIO::Elf64\_Addr>::iterator it;

for (it=result.begin(); it!=result.end(); ++it)

{

stub.set(it->second + base\_addr ,xxx\_stub);

std::cout << it->first << " => " << it->second + base\_addr<<std::endl;

}

test\_test();

return 0;

}

**说明：**

* 只适用linux，和windows的x86、x64架构

**不可以打桩的情况：**

* 不可以对exit函数打桩，编译器做了特殊优化
* 不可以对纯虚函数打桩，纯虚函数没有地址

**linux下单元测试一些编译选项：**

