

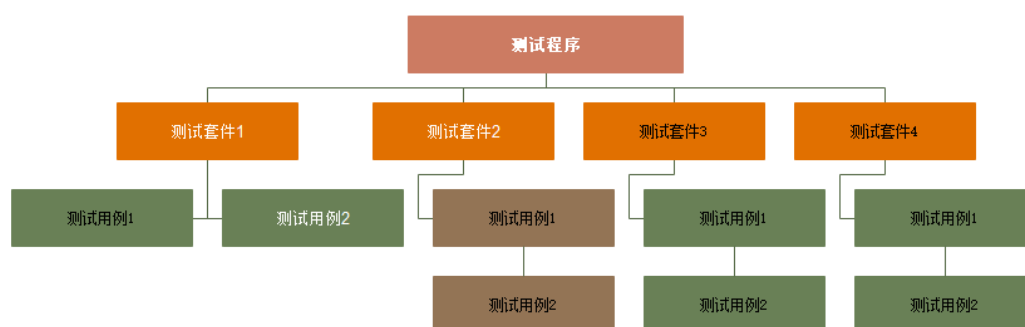
gtest 简介

gtest 是一个跨平台的(Linux、Mac OS X、Windows、Cygwin、Windows CE and Symbian)C++单元测试框架，由 google 公司发布。gtest 是为在不同平台上为编写 C++测试而生成的。它提供了丰富的断言、致命和非致命判断、参数化、“死亡测试”等等。

gtest 原理

gtest 主要由一系列的宏和事件实现。

1. 宏：有 TEST 和 TEST_F 宏，TEST 宏针对简单的测试用例，TEST_F 宏针对需要做初始化和资源回收的测试用例，有点像类似 C++ 的构造函数和析构函数，两个宏都是把参数展开后拼成一个类。
2. 事件：分为三种事件



测试程序：一个进程，全局事件在该层

测试套件：一系列测试用例的集合，SetUpTestCase 事件在该层

测试用例：一个测试用例的事件体现在每个用例前后的 SetUp 和 TearDown

每个事件可用于下级事件的数据共享以及测试前后的数据处理

gtest 事件机制

“事件”本质是 gtest 框架提供了一个机会，可以使用这样的机会来执行定制的代码，来给测试用例准备/清理数据。gtest 提供了多种事件机制，总结一下 gtest 的事件一共有三种：

1、TestSuite 事件

需要写一个类，继承 testing::Test，然后实现两个静态方法：**SetUpTestCase** 方法在第一个 **TestCase** 之前执行；**TearDownTestCase** 方法在最后一个 **TestCase** 之后执行。

如下是一个例子：

```

1  #ifndef MYGTEST_H
2  #define MYGTEST_H
3  #include <gtest/gtest.h>
4  #include "MyTest.h"
5
6  using namespace testing;
7
8  class MyGTest : public Test
9  {
10 public:
11     void SetUp();
12     void TearDown();
13
14 protected:
15     MyTest* myTest;
16 };
17
18 #endif // MYGTEST_H

```

```

1  #include "MyGTest.h"
2
3  void MyGTest::SetUp(){
4      myTest = new MyTest();
5  }
6
7  void MyGTest::TearDown(){
8      if (myTest != nullptr){
9          delete myTest;
10         myTest = nullptr;
11     }
12 }
13
14 TEST_F(MyGTest, setParameterTest){
15     myTest->setParameter(10, 20);
16     EXPECT_EQ(10, myTest->getA());
17     EXPECT_EQ(20, myTest->getB());
18 }
19
20 TEST_F(MyGTest, getSumTest){
21     myTest->setParameter(1,5);
22     auto Ret = myTest->getSum();
23     EXPECT_EQ(6, Ret);
24 }

```

运行结果如下:

```

===== Running 2 tests from 1 test case.
----- Global test environment set-up.
----- 2 tests from MyGTest
RUN      MyGTest.setParameterTest
          OK      MyGTest.setParameterTest (0 ms)
RUN      MyGTest.getSumTest
          OK      MyGTest.getSumTest (0 ms)
----- 2 tests from MyGTest (0 ms total)

----- Global test environment tear-down
===== 2 tests from 1 test case ran. (0 ms total)
PASSED  2 tests.

```

2、TestCase 事件

是挂在每个用例执行前后的，需要实现的是 SetUp 方法和 TearDown 方法。SetUp 方法在每个 TestCase 之前执行；TearDown 方法在每个 TestCase 之后执行。

在 1 中的 SetUp 和 TearDown 中添加打印语句：

```
MyGTest.cpp
1 #include "MyGTest.h"
2
3 void MyGTest::SetUp(){
4     cout << "MyGTest::SetUp called...\n";
5     myTest = new MyTest();
6 }
7
8 void MyGTest::TearDown(){
9     cout << "MyGTest::TearDown called...\n";
10    if (myTest != nullptr){
11        delete myTest;
12        myTest = nullptr;
13    }
14 }
15
16 TEST_F(MyGTest, setParameterTest){
17     myTest->setParameter(10, 20);
18     EXPECT_EQ(10, myTest->getA());
19     EXPECT_EQ(20, myTest->getB());
20 }
21
22 TEST_F(MyGTest, getSumTest){
23     myTest->setParameter(1,5);
24     auto Ret = myTest->getSum();
25     EXPECT_EQ(6, Ret);
26 }
```

```
Terminal
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Running 2 tests from 1 test case.
Global test environment set-up.
2 tests from MyGTest
RUN MyGTest.setParameterTest
MyGTest::SetUp called...
MyGTest::TearDown called...
OK MyGTest.setParameterTest (0 ms)
RUN MyGTest.getSumTest
MyGTest::SetUp called...
MyGTest::TearDown called...
OK MyGTest.getSumTest (0 ms)
2 tests from MyGTest (0 ms total)
Global test environment tear-down
2 tests from 1 test case ran. (0 ms total)
PASSED 2 tests.
按 <RETURN> 来关闭窗口...
```

3、全局事件

要实现全局事件，必须写一个类，继承 `testing::Environment` 类，实现里面的 `SetUp` 和 `TearDown` 方法。**SetUp 方法在所有测试用例执行前执行；TearDown 方法在所有测试用例执行后执行。**

除了要继承 `testing::Environment` 类，还要定义一个该全局环境的一个对象并将该对象添加到全局测试环境中。

全局事件可以按照下列方式来使用：

```
class GlobalTest : public testing::Environment {
public:
    virtual void SetUp() {
        cout << "excute before" << endl;
    }

    virtual void TearDown() {
        cout << "excute after" << endl;
    }
};

int abs(int x) {
    return x > 0 ? x : -x;
}

int add(int a, int b) {
    return a + b;
}

TEST(GlobalTest, abs) {
    ASSERT_EQ(abs(1), abs(-1));
}

TEST(GlobalTest, add) {
    ASSERT_TRUE(add(2, 3) == 4);
}

int main(int argc, char *argv[]) {
    testing::InitGoogleTest(&argc, argv); //将命令行参数传递给gtest

    // 下面这两句代码是使用全局事件必备的
    testing::Environment *env = new GlobalTest();
    testing::AddGlobalTestEnvironment(env);

    return RUN_ALL_TESTS(); // RUN_ALL_TESTS()运行所有测试用例
}
```

```
Terminal
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
Running 2 tests from 1 test case.
Global test environment set-up.
excute before
2 tests from GlobalTest
RUN GlobalTest.abs
OK GlobalTest.abs (0 ms)
RUN GlobalTest.add
../gtest-demo/main.cpp:30: Failure
Value of: add(2, 3) == 4
Actual: false
Expected: true
FAILED GlobalTest.add (0 ms)
2 tests from GlobalTest (0 ms total)
Global test environment tear-down
excute after
2 tests from 1 test case ran. (0 ms total)
1 test.
FAILED 1 test, listed below:
FAILED GlobalTest.add
1 FAILED TEST
按 <RETURN> 来关闭窗口...
```

gtest 内置的宏

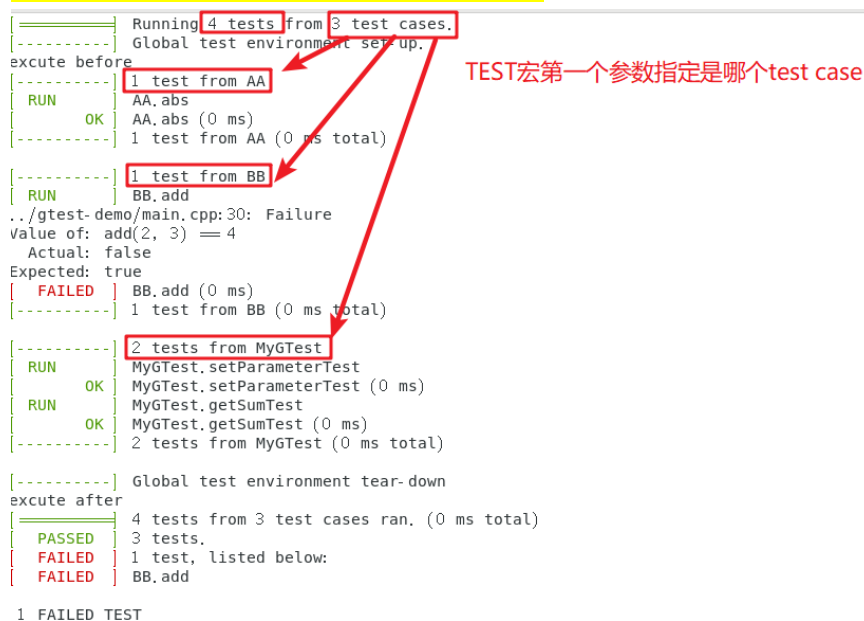
TEST(test_case_name, test_name)

TEST_F(test_fixture, test_name) //多个测试场景需要相同数据配置的情况, 用TEST_F。TEST_F test fixture, 测试夹具, 测试套, 承担了一个注册的功能 (为测试类的类名)。

TEST_F 宏的使用需要按照 C++ 的风格定义类, 这个类需要继承自 `testing::Test` 并

且重写这四个成员函数: `SetUpTestCase`、`TearDownTestCase`、`SetUp`、`TearDown`。如果有多个测试用例, 那么 `SetUpTestCase` 会在所有用例之前执行, `TearDownTestCase` 会在所有用例之后执行; `SetUp` 会在单个用例之前执行, `TearDown` 会在单个用例之后执行。

【注意】使用 TEST 宏时, test_case_name 不要求是当前测试类的类名, 而使用 TEST_F 宏时, 第一个参数必须是当前测试类的类名。



```
Running 4 tests from 3 test cases.
Global test environment set-up.
[-----]
[ RUN      ] 1 test from AA
[ OK       ] AA.abs
[ OK       ] AA.abs (0 ms)
[-----] 1 test from AA (0 ms total)

[-----]
[ RUN      ] 1 test from BB
[ OK       ] BB.add
[ FAIL     ] ..gtest-demo/main.cpp:30: Failure
Value of: add(2, 3) == 4
Actual: false
Expected: true
[ FAILED   ] BB.add (0 ms)
[-----] 1 test from BB (0 ms total)

[-----]
[ RUN      ] 2 tests from MyGTest
[ OK       ] MyGTest.setParameterTest
[ OK       ] MyGTest.setParameterTest (0 ms)
[ RUN      ] MyGTest.getSumTest
[ OK       ] MyGTest.getSumTest (0 ms)
[-----] 2 tests from MyGTest (0 ms total)

[-----] Global test environment tear-down
[ OK       ]
[-----]
[ PASSED   ] 4 tests from 3 test cases ran. (0 ms total)
[ PASSED   ] 3 tests.
[ FAILED   ] 1 test, listed below:
[ FAILED   ] BB.add

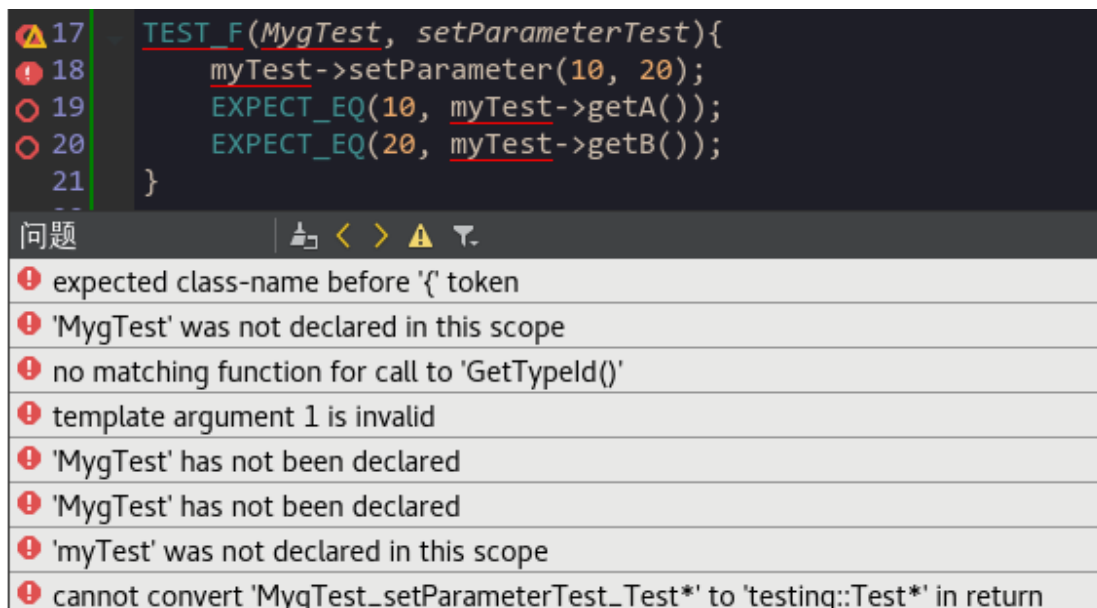
1 FAILED TEST
```



```
gtest-demo/main.cpp
1 #include <iostream>
2 #include <gtest/gtest.h>
3 #include "MyGTest.h"
4
5 using namespace std;
6
7 int abs(int x) {
8     return x > 0 ? x : -x;
9 }
10
11 TEST(abs_test, abs) {
12     ASSERT_TRUE(abs(1) != 1);
13 }

MyGTest.cpp
1 #include "MyGTest.h"
2
3 void MyGTest::SetUp() {
4     myTest = new MyTest();
5 }
6
7 void MyGTest::TearDown() {
8     if (myTest != nullptr) {
9         delete myTest;
10        myTest = nullptr;
11    }
12 }
13
14 TEST_F(MyGTest, setParameterTest) {
15     myTest->setParameter(10, 20);
16     EXPECT_EQ(10, myTest->getA());
17     EXPECT_EQ(20, myTest->getB());
18 }
19
20 TEST_F(MyGTest, getSumTest) {
21     myTest->setParameter(1, 5);
22     auto Ret = myTest->getSum();
23     EXPECT_EQ(6, Ret);
24 }
```

TEST_F 宏第一个参数不是当前测试类的类名时, 编译错误:



TEST 宏的作用是创建一个简单测试，它定义了一个测试函数，在这个函数里可以使用任何 C++ 代码并使用提供的断言来进行检查。

gtest 单元测试断言

gtest 中断言的宏可以分为两类：一类是 ASSERT 宏，另一类是 EXPECT 宏。

- 1、ASSERT_系列：如果当前点检测失败则退出当前函数
- 2、EXPECT_系列：如果当前点检测失败则继续往下执行

如果对自动输出的错误信息不满意的话，也可以通过 operator<< 在失败的时候打印日志，将一些自定义的信息输出。

常用 ASSERT_系列：

宏类型	宏名称	功能	描述
bool 值 检查	ASSERT_TRUE(条件)	判断条件是否为 true	期待结果是 true
	ASSERT_FALSE(条件)	判断条件是否为 false	期待结果是 false
数值型 数据检 查	ASSERT_EQ(参数 1, 参数 2)	equal	相等为 true
	ASSERT_NE(参数 1, 参数 2)	not equal	不等于返回 true
	ASSERT_LT(参数 1, 参数 2)	less than	小于返回 true
	ASSERT_GT(参数 1, 参数 2)	greater than	大于返回 true
	ASSERT_LE(参数 1, 参数 2)	less equal	小于等于返回 true
	ASSERT_GE(参数 1, 参数 2)	greater equal	大于等于返回 true
	ASSERT_FLOAT_EQ(expected, actual)	equal	两个 float 值相同返回 true
	ASSERT_DOUBLE_EQ(expected, actual)	equal	两个 double 值相同返回 true
字符串 检查	ASSERT_STREQ(expected_str, actual_str)	判断字符串是否相等	两个 C 风格的字符串相等返回 true

	ASSERT_STRNE(expected, actual)	判断字符串不相等	两个 C 风格的字符串不相等时返回 true
	ASSERT_STRCASEEQ(expected_str, actual_str)	判断字符串是否相等(忽略大小写)	
	ASSERT_STRCASENE(str1, str2)	判断字符串不相等(忽略大小写)	

EXPECT_系列，具有 ASSERT 系列类似的宏结构。

如下是自定义信息输出：

```
TEST(GlobalTest, add) {
    ASSERT_TRUE(add(2, 3) == 4) << "Expect 5, but given 4";
}

int main(int argc, char *argv[]) {
    RUN(GlobalTest.add)
    ../gtest-demo/main.cpp:30: Failure
    Value of: add(2, 3) == 4
    Actual: false
    Expected: true
    Expect 5, but given 4
    FAILED | GlobalTest.add (0 ms)
```

gtest 安装

1、更新 dnf 源

sudo yum install epel-release

```
[root@VM ~]# yum install epel-release
已加载插件: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
epel/x86_64/metalink
 * base: mirrors.aliyun.com
 * epel: mirrors.bfsu.edu.cn
 * extras: mirrors.aliyun.com
 * updates: mirrors.aliyun.com
base                               | 3.6 kB 00:00:00
epel                               | 4.7 kB 00:00:00
extras                             | 2.9 kB 00:00:00
google-chrome                      | 1.3 kB 00:00:00
updates                            | 2.9 kB 00:00:00
(1/4): epel/x86_64/updateinfo      | 1.0 MB 00:00:10
(2/4): epel/x86_64/primary_db      | 7.0 MB 00:00:00
(3/4): google-chrome/primary       | 1.7 kB 00:00:05
(4/4): updates/7/x86_64/primary_db | 15 MB 00:01:33
3/3
正在解决依赖关系
--> 正在检查事务
--> 软件包 epel-release.noarch.0.7-12 将被 升级
--> 软件包 epel-release.noarch.0.7-14 将被 更新
--> 解决依赖关系完成

依赖关系解决
=====
Package                        架构      版本      源      大小
正在更新:
epel-release                   noarch    7-14      epel    15 k
=====
事务概要
-----
升级   1 软件包

总计: 15 k
Is this ok [y/d/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
 正在更新   : epel-release-7-14.noarch
 清理       : epel-release-7-12.noarch
 验证中     : epel-release-7-14.noarch
 验证中     : epel-release-7-12.noarch

更新完毕:
epel-release.noarch 0:7-14

完毕！
```

sudo yum install dnf


```
[root@VM ~]# yum install dnf
已加载插件: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirrors.aliyun.com
 * epel: mirrors.bfsu.edu.cn
 * extras: mirrors.aliyun.com
 * updates: mirrors.aliyun.com
正在解决依赖关系
--> 正在检查事务
--> 软件包 dnf.noarch.0.4.0.9.2-2.el7_9 将被 安装
--> 正在处理依赖关系 python2-dnf = 4.0.9.2-2.el7_9, 它被软件包 dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在检查事务
--> 软件包 python2-dnf.noarch.0.4.0.9.2-2.el7_9 将被 安装
--> 正在处理依赖关系 dnf-data = 4.0.9.2-2.el7_9, 它被软件包 python2-dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在处理依赖关系 python2-libdnf >= 0.22.5, 它被软件包 python2-dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在处理依赖关系 python2-libcomps >= 0.1.8, 它被软件包 python2-dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在处理依赖关系 python2-hawkey >= 0.22.5, 它被软件包 python2-dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在处理依赖关系 libmodulemd >= 1.4.0, 它被软件包 python2-dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在处理依赖关系 python2-libdnf, 它被软件包 python2-dnf-4.0.9.2-2.el7_9.noarch 需要
--> 正在检查事务
--> 软件包 dnf-data.noarch.0.4.0.9.2-2.el7_9 将被 安装
--> 软件包 libmodulemd.x86_64.0.1.6.3-1.el7 将被 安装
--> 软件包 python2-hawkey.x86_64.0.22.5-2.el7_9 将被 安装
--> 正在处理依赖关系 libdnf(x86-64) = 0.22.5-2.el7_9, 它被软件包 python2-hawkey-0.22.5-2.el7_9.x86_64 需要
--> 正在处理依赖关系 libsolvext.so.0(SOLV 1.0)(64bit), 它被软件包 python2-hawkey-0.22.5-2.el7_9.x86_64 需要
--> 正在处理依赖关系 libsolv.so.0(SOLV 1.0)(64bit), 它被软件包 python2-hawkey-0.22.5-2.el7_9.x86_64 需要
--> 正在处理依赖关系 libsolvext.so.0()(64bit), 它被软件包 python2-hawkey-0.22.5-2.el7_9.x86_64 需要
--> 正在处理依赖关系 libsolv.so.0()(64bit), 它被软件包 python2-hawkey-0.22.5-2.el7_9.x86_64 需要
--> 正在处理依赖关系 librepo.so.0()(64bit), 它被软件包 python2-hawkey-0.22.5-2.el7_9.x86_64 需要
```

```
总计
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
 正在安装      : libmodulemd-1.6.3-1.el7.x86_64
 正在安装      : libsolv-0.6.34-4.el7.x86_64
 正在安装      : librepo-1.8.1-8.el7_9.x86_64
 正在安装      : libdnf-0.22.5-2.el7_9.x86_64
 正在安装      : python2-libdnf-0.22.5-2.el7_9.x86_64
 正在安装      : python2-hawkey-0.22.5-2.el7_9.x86_64
 正在安装      : libcomps-0.1.8-14.el7.x86_64
 正在安装      : python2-libcomps-0.1.8-14.el7.x86_64
 正在安装      : dnf-data-4.0.9.2-2.el7_9.noarch
 正在安装      : python2-dnf-4.0.9.2-2.el7_9.noarch
 正在安装      : dnf-4.0.9.2-2.el7_9.noarch
 验证中       : python2-libcomps-0.1.8-14.el7.x86_64
 验证中       : dnf-4.0.9.2-2.el7_9.noarch
 验证中       : librepo-1.8.1-8.el7_9.x86_64
 验证中       : python2-hawkey-0.22.5-2.el7_9.x86_64
 验证中       : libmodulemd-1.6.3-1.el7.x86_64
 验证中       : dnf-data-4.0.9.2-2.el7_9.noarch
 验证中       : libdnf-0.22.5-2.el7_9.x86_64
 验证中       : python2-dnf-4.0.9.2-2.el7_9.noarch
 验证中       : libcomps-0.1.8-14.el7.x86_64
 验证中       : libsolv-0.6.34-4.el7.x86_64
 验证中       : python2-libdnf-0.22.5-2.el7_9.x86_64

已安装:
  dnf.noarch 0:4.0.9.2-2.el7_9

作为依赖被安装:
  dnf-data.noarch 0:4.0.9.2-2.el7_9      libcomps.x86_64 0:
  python2-libcomps.x86_64 0:0.1.8-14.el7 python2-libdnf.x86

完毕!
```

2、安装 gtest

```
sudo dnf install dnf-plugins-core  
sudo dnf install gtest gtest-devel
```

```
[root@VM ~]# dnf install dnf-plugins-core  
Extra Packages for Enterprise Linux 7 - x86_64  
google-chrome  
CentOS-7 - Base - mirrors.aliyun.com  
CentOS-7 - Updates - mirrors.aliyun.com  
CentOS-7 - Extras - mirrors.aliyun.com  
依赖关系解决。  
=====
```

软件包
dnf-plugins-core

```
Installing:  
  dnf-plugins-core  
安装依赖关系:  
  python2-dnf-plugins-core  
事务概要  
=====
```

安装 2 软件包
总下载: 216 k
安装大小: 546 k
确定吗? [y/N]: y
下载软件包:
(1/2): python2-dnf-plugins-core-4.0.2.2-3.el7_6.noarch.rpm
(2/2): dnf-plugins-core-4.0.2.2-3.el7_6.noarch.rpm

```
-----  
总计  
运行事务检查  
事务检查成功。  
运行事务测试  
事务测试成功。  
运行事务  
  准备中      :  
Installing   : python2-dnf-plugins-core-4.0.2.2-3.el7_6.noarch  
Installing   : dnf-plugins-core-4.0.2.2-3.el7_6.noarch  
验证         : dnf-plugins-core-4.0.2.2-3.el7_6.noarch  
验证         : python2-dnf-plugins-core-4.0.2.2-3.el7_6.noarch  
已安装:  
  dnf-plugins-core-4.0.2.2-3.el7_6.noarch  
完毕!
```

```
[root@VM ~]# dnf install gtest gtest-devel  
上次元数据过期检查: 0:02:16 前, 执行于 2022年04月18日 星期一 11时27分07秒。  
依赖关系解决。  
=====
```

已安装:
gtest-1.6.0-2.el7.x86_64
gtest-devel-1.6.0-2.el7.x86_64

```
完毕!  
root@VM ~]# find /usr -name gtest.h  
usr/include/gtest/gtest.h  
root@VM ~]#
```


gtest-demo

简单用法

```
1  #include <iostream>
2  #include <gtest/gtest.h>
3
4  using namespace std;
5
6  int abs(int x) {
7      return x > 0 ? x : -x;
8  }
9
10 TEST(abs_test, abs) {
11     ASSERT_TRUE(abs(1) != 1);
12 }
13
14 int main(int argc, char *argv[]) {
15     testing::InitGoogleTest(&argc, argv); //将命令行参数传递给gtest
16     return RUN_ALL_TESTS(); // RUN_ALL_TESTS()运行所有测试用例
17 }
18
```

```
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from abs_test
[ RUN      ] abs_test.abs ← 模块名称
main.cpp:12: Failure
Value of: abs(1) != 1 ← 测试语句
Actual: false
Expected: true ← 实际值和期望值，两个值不同，所以FAILED
[ FAILED   ] abs_test.abs (0 ms)
[-----] 1 test from abs_test (0 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test case ran. (0 ms total)
[ PASSED   ] 0 tests.
[ FAILED   ] 1 test, listed below:
[ FAILED   ] abs_test.abs

1 FAILED TEST
```

作为测试类使用

1、定义一个要被测试的类

```
class MyTest
{
public:
    MyTest();

    void setParameter(int p1,int p2);
    unsigned int getSum();
    int getA();
    int getB();

private:
    int a;
    int b;
};
```

```
#include "MyTest.h"

MyTest::MyTest()
{
    a = 0;
    b = 0;
}

void MyTest::setParameter(int p1, int p2){
    a = p1;
    b = p2;
}

unsigned int MyTest::getSum(){
    return static_cast<unsigned int>(a+b);
}

int MyTest::getA(){
    return a;
}

int MyTest::getB(){
    return b;
}
```

2、定义测试类

把要测试的类作为测试类的成员变量, 重写 SetUpTestCase、TearDownTestCase、SetUp、TearDown 成员函数。

```
void MyGTest::SetUp(){
    cout << "MyGTest::SetUp called...\n";
    myTest = new MyTest();
}

void MyGTest::TearDown(){
    cout << "MyGTest::TearDown called...\n";
    if (myTest != nullptr){
        delete myTest;
        myTest = nullptr;
    }
}

void MyGTest::SetUpTestCase(){
    cout << "MyGTest::SetUpTestCase called...\n";
}

void MyGTest::TearDownTestCase(){
    cout << "MyGTest::TearDownTestCase called...\n";
}

TEST_F(MyGTest, setParameterTest){
    myTest->setParameter(10, 20);
    EXPECT_EQ(10, myTest->getA());
    EXPECT_EQ(20, myTest->getB());
}

TEST_F(MyGTest, getSumTest){
    myTest->setParameter(1,5);
    auto Ret = myTest->getSum();
    EXPECT_EQ(6, Ret);
}
```

```
[-----] 2 tests from MyGTest
MyGTest::SetUpTestCase called...
RUN      MyGTest.setParameterTest
MyGTest::SetUp called...
MyGTest::TearDown called...
OK       MyGTest.setParameterTest (0 ms)
RUN      MyGTest.getSumTest
MyGTest::SetUp called...
MyGTest::TearDown called...
OK       MyGTest.getSumTest (0 ms)
MyGTest::TearDownTestCase called...
[-----] 2 tests from MyGTest (0 ms total)
```

实测 `SetUpTestCase`、`TearDownTestCase`、`SetUp`、`TearDown` 四个函数只用重写其中的一对即可(如 `SetUp` 和 `TearDown`、`SetUpTestCase` 和 `TearDownTestCase` 均可)。可以按实际需求选择要重写的函数。注意 `SetUpTestCase` 和 `TearDownTestCase` 是 static 函数，`SetUp` 和 `TearDown` 是虚函数。

```
class GTEST_API_ Test {
public:
    friend class TestInfo;

    // Defines types for pointers to functions that set up and tear down
    // a test case.
    typedef internal::SetUpTestCaseFunc SetUpTestCaseFunc;
    typedef internal::TearDownTestCaseFunc TearDownTestCaseFunc;

    // The d'tor is virtual as we intend to inherit from Test.
    virtual ~Test();

    // Sets up the stuff shared by all tests in this test case.
    //
    // Google Test will call Foo::SetUpTestCase() before running the first
    // test in test case Foo. Hence a sub-class can define its own
    // SetUpTestCase() method to shadow the one defined in the super
    // class.
    static void SetUpTestCase() {}

    // Tears down the stuff shared by all tests in this test case.
    //
    // Google Test will call Foo::TearDownTestCase() after running the last
    // test in test case Foo. Hence a sub-class can define its own
    // TearDownTestCase() method to shadow the one defined in the super
    // class.
    static void TearDownTestCase() {}

    // Sets up the test fixture.
    virtual void SetUp();

    // Tears down the test fixture.
    virtual void TearDown();
};
```

重写任意一对即可：

<pre>void MyGTest::SetUp(){ cout << "MyGTest::SetUp called...\n"; myTest = new MyTest(); } void MyGTest::TearDown(){ cout << "MyGTest::TearDown called...\n"; if (myTest != nullptr){ delete myTest; myTest = nullptr; } }</pre>	<pre>[-----] 2 tests from MyGTest [RUN] MyGTest.setParameterTest MyGTest::SetUp called... MyGTest::TearDown called... [OK] MyGTest.setParameterTest (0 ms) [RUN] MyGTest.getSumTest MyGTest::SetUp called... MyGTest::TearDown called... [OK] MyGTest.getSumTest (0 ms) [-----] 2 tests from MyGTest (0 ms total)</pre>
<pre>void MyGTest::SetUpTestCase(){ cout << "MyGTest::SetUpTestCase called...\n"; myTest = new MyTest(); } void MyGTest::TearDownTestCase(){ cout << "MyGTest::TearDownTestCase called...\n"; if (myTest != nullptr){ delete myTest; myTest = nullptr; } }</pre>	<pre>[-----] Global test environment tear-down [FAILED] GlobalTest.add (0 ms) [-----] 2 tests from GlobalTest (0 ms total) [-----] 2 tests from MyGTest MyGTest::SetUpTestCase called... [RUN] MyGTest.setParameterTest [OK] MyGTest.setParameterTest (0 ms) [RUN] MyGTest.getSumTest [OK] MyGTest.getSumTest (0 ms) MyGTest::TearDownTestCase called... [-----] 2 tests from MyGTest (0 ms total) [-----] Global test environment tear-down</pre>

3、执行所有用例

```
int main(int argc, char *argv[]) {
    testing::InitGoogleTest(&argc, argv); //将命令行参数传递给gtest

    // 下面这两句代码是使用全局事件必备的
    testing::Environment *env = new GlobalTest();
    testing::AddGlobalTestEnvironment(env);

    return RUN_ALL_TESTS(); // RUN_ALL_TESTS()运行所有测试用例
}
```

```
[=====] Running 7 tests from 2 test cases.
[-----] Global test environment set-up.
excute before
[-----] 2 tests from GlobalTest
[ RUN      ] GlobalTest.abs
[          OK ] GlobalTest.abs (0 ms)
[ RUN      ] GlobalTest.add
../gtest-demo/main.cpp:35: Failure
Value of: add(2, 3) == 4
Actual: false
Expected: true
Expect 5, but given 4
[   FAILED   ] GlobalTest.add (0 ms)
[-----] 2 tests from GlobalTest (0 ms total)

[-----] 5 tests from MyGTest
MyGTest::SetUpTestCase called...
[ RUN      ] MyGTest.setParameterTest
[          OK ] MyGTest.setParameterTest (0 ms)
[ RUN      ] MyGTest.getSumTest
[          OK ] MyGTest.getSumTest (0 ms)
[ RUN      ] MyGTest.strCaseTest
[          OK ] MyGTest.strCaseTest (0 ms)
[ RUN      ] MyGTest.Case1
[          OK ] MyGTest.Case1 (0 ms)
[ RUN      ] MyGTest.Case2
[          OK ] MyGTest.Case2 (0 ms)
MyGTest::TearDownTestCase called...
[-----] 5 tests from MyGTest (0 ms total)

[-----] Global test environment tear-down
excute after
[=====] 7 tests from 2 test cases ran. (0 ms total)
[ PASSED    ] 6 tests.
[   FAILED   ] 1 test, listed below:
[   FAILED   ] GlobalTest.add
```

1 FAILED TEST