Google Test

一、简单介绍

gtest 是一个跨平台的 C++单元测试框架,提供了丰富的断言致命和非致命判断、参数化等等。

特点:

1. 跨平台;
2. 使用简便;
3. 打印信息按测试案例分类，可直接定位位置，
4. 返回必要参数并可以尾部追加信息
5. 同时简单设置后，直接输出 xml 报告;

二、下载地址

https://github.com/google/googletest/tree/master/googletest

三、基本使用

我们的测试案例通过 define TEST(test_case_name, test_name)宏实现。

TEST(test_case_name, test_name){

调用对应接口

}

在主函数中做简单的初始化工作

testing::InitGoogleTest(&argc, argv);

RUN_ALL_TESTS();

1）断言

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

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

常见有布尔值检查，数值型数据检查，字符串检查，异常检查（在 gtest.h 中有详细说明）

范例：

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

```
TEST(ADD_Test) {
    EXPECT_EQ(14, add(4, 10));
    EXPECT_EQ(39, add(30, 18));
}
```

```
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from ADD_Test
[ RUN      ] ADD_Test.
e:\googoltest\consoleapplication1\consoleapplication1\consoleapplication1.cpp(35): error: Value of: add(30, 18)
  Actual: 48
Expected: 39
[  FAILED  ] ADD_Test. (1 ms)
[----------] 1 test from ADD_Test (3 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (6 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] ADD_Test.
```

2）事件

这里的事件指的是测试数案例前和测试按钮后必定触发的函数。一般初始化和释放内存等操作。

事件分 3 种：

1. 在案例集之前和案例集后触发（全局）
2. 在选择的某一类测试案例执行前和执行后触发
3. 在选择的某一个测试案例执行前和执行后触发

事件 1

1. 继承 testing::Environment
2. 重载 SetUp()和 TearDown()
3. 主函数中初始化
   `testing::AddGlobalTestEnvironment(object);`

示例

```
class GrobalEvent : public testing::Environment
{
  public:
    virtual void SetUp(){}
    virtual void TearDown(){}
};
```

事件 2

1. 继承 testing::Test
2. 重载 SetUpTestCase()和 TearDownTestCase()并设置为静态
3. 测试案例用到的宏由 TEST 改成 TEST_F,并输入类名

示例

```
class FooTest : public testing::Test
```

```
{
  protected:
    static void SetUpTestCase(){}
    static void TearDownTestCase(){}
};

TEST_F(FooTest,ZeroEqual)
{
}
TEST_F(FooTest,OneEqual)
{
}
```

事件 3

1. 继承 testing::Tes
2. t 重载 SetUp()和 TearDown()
3. 测试案例用到的宏由 TEST 改成 TEST_F,并输入类名

示例

```
class TestCase : public testing::Test
{
     protected:
       virtual void SetUp(){}
       virtual void TearDown(){}
};
TEST_F(TestCase,check1)
{
}
TEST_F(TestCase,check2)
{
}
```

3）多参数测试

根据之前断言和 TEST()的使用方法，我们编写测试案例时会有大量的冗余代码：

如

```
bool IsPrime(int n);
TEST(IsPrimeTest, Negative)
{
  EXPECT_FALSE(IsPrime(-1));
  EXPECT_FALSE(IsPrime(-2));
  EXPECT_FALSE(IsPrime(-5));
  EXPECT_FALSE(IsPrime(-100));
  EXPECT_FALSE(IsPrime(INT_MIN));
}
```

对“EXPECT_FALSE(IsPrime(X))”这样的语句复制粘贴了 5 次，但如果要测试的数据上千个或者根本就不知道要测试多少个数据，显然，以上方案不合理。因此在 gtest 中提供了一种方法“多参数测试”来解决以上问题。

使用方式：

1. 继承 testing::TestWithParam<int>
2. 定义一个变量并在构造函数中使用 GetParam()获取后续传递的值
3. 测试案例的宏由 TEST 改成 TEST_P，并传递类名
4. 使用 INSTANTIATE_TEST_CASE_P 传递所需的参数（可配合定时器和多线程）

示例

```
class Multi_Param_TestCase : public
testing::TestWithParam<int>
{
   public:
    Multi_Param_TestCase(){m_n = GetParam();}
   protected:
    int m_n;
};

TEST_P(Multi_Param_TestCase, HandleTureReturn)
{
EXPECT_FALSE(IsPrime(m_n));
}
INSTANTIATE_TEST_CASE_P(NaiveAndFast,
                Multi_Param_TestCase,
                testing::Values(1,3,4,5,7,9,10));
```