Magellan

A Simple xUnit Test Framework in Modern C++11

刘光聪

This page is intentionally left blank.

目录

This page is intentionally left blank.

I'm not a great programmer; I'm just a good programmer with great habits.

- Kent Beck

1 初识 Magellan

1.1 简介

Magellan 是一个简单的、可扩展的、使用 C++11 实现的 xUnit 测试框架，Magellan 设计灵感来自于 Java 社区著名的测试框架 JUnit。

1.2 安装

1.2.1 环境准备

编译环境

Magellan 目前仅在 Linux, MAC OS X 系统上测试过，仅支持 GCC，CLANG 两款编译器。

编译器	最低版本
GCC	4.8
CLANG	3.4

表 1.1 支持编译器及其版本

安装 CMake

CMake 的下载地址是：http://www.cmake.org。以 Ubuntu 为例，使用 apt-get 安装 CMake。

示例代码 1-1 安装 CMAKE

```
$ sudo apt-get install cmake
```

安装 l0-infra

Magellan 依赖于 l0-infra，所以必须先安装 l0-infra

示例代码 1-2 安装 RVM

```
$ git clone https://gitlab.com/horance/l0-infra.git
$ cd l0-infra
$ mkdir build
$ cd build
```

```
$ cmake ..
$ make
$ sudo make install
```

1.2.2 安装 Magellan

示例代码 1-3 安装 RVM

```
$ git clone https://gitlab.com/horance/magellan.git
$ cd magellan
$ mkdir build
$ cd build
$ cmake ..
$ make
$ sudo make install
```

1.3 破冰之旅

1.3.1 物理目录

示例代码 1-4 Quantity 物理设计

```
--quantity
  |-- include
  | |-- quantity
  |-- src
  | |-- quantity
  |-- test
  | |-- quantity
  | |-- main.cpp
  |-- CmakeList.txt
```

1.3.2 main 函数

示例代码 1-5 test/main.cpp

```
#include "magellan/magellan.hpp"

int main(int argc, char** argv)
{
    return magellan::run_all_tests(argc, argv);
}
```

1.3.3 CMAKE 构建脚本

示例代码 1-6 quantity/CMakeLists.txt

```
project(quantity)

cmake_minimum_required(VERSION 2.8)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++0x")

include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include)

file(GLOB_RECURSE all_files
src/*.cpp
src/*.cc
src/*.c
test/*.cpp
test/*.cc
test/*.c)

add_executable(quantity-test ${all_files})

target_link_libraries(quantity-test magellan l0-infra)
```

1.3.4 构建 Quantity

示例代码 1-7 构建 Quantity

```
$ mkdir build
$ cd build
$ cmake ..
$ make
```

1.3.5 执行测试

示例代码 1-8 执行测试

```
$ ./quantity-test

[==========] Running 0 test cases.
[----------] 0 tests from All Tests
[----------] 0 tests from All Tests

[==========] 0 test cases ran.
[  TOTAL   ] PASS: 0  FAILURE: 0  ERROR: 0  TIME: 0 us
```


1.4 体验 Magellan

1.4.1 第一个测试用例

示例代码 1-9 test/quantity/LengthTest.cpp

```
#include <magellan/magellan.hpp>
#include "quantity/Length.h"

USING_HAMCREST_NS

FIXTURE(LengthTest)
{
    TEST("1 FEET should equal to 12 INCH")
    {
        ASSERT_THAT(Length(1, FEET), eq(Length(12, INCH)));
    }
};
```

使用 Magellan，只需要包含 magellan.hpp 一个头文件即可。Magellan 使用 Hamcrest 的断言机制，使得断言更加统一、自然，且具有良好的扩展性；使用 USING_HAMCREST_NS，从而可以使用 eq 代替 hamcrest::eq，简短明确；除非出现名字冲突，否则推荐使用简写的 Matcher。

1.4.2 Length 实现

示例代码 1-10 include/quantity/Length.h

```
#include "quantity/Amount.h"

enum LengthUnit
{
    INCH = 1,
    FEET = 12 * INCH,
};

struct Length
{
    Length(Amount amount, LengthUnit unit);

    bool operator==(const Length&) const;
    bool operator!=(const Length&) const;

private:
    const Amount amountInBaseUnit;
};
```

示例代码 1-11 src/quantity/Length.cpp

```
#include "quantity/Length.h"

Length::Length(Amount amount, LengthUnit unit)
  : amountInBaseUnit(unit * amount)
{
}

bool Length::operator==(const Length& rhs) const
{
    return amountInBaseUnit == rhs.amountInBaseUnit;
}

bool Length::operator!=(const Length& rhs) const
{
    return !(*this == rhs);
}
```

1.4.3 构建 Quantity

示例代码 1-12 构建 Quantity

```
$ cd build
$ cmake ..
$ make
```

1.4.4 执行测试

示例代码 1-13 执行测试

```
$ ./quantity-test

[==========] Running 1 test cases.
[----------] 1 tests from All Tests
[----------] 1 tests from LengthTest
[ RUN      ] LengthTest::1 FEET should equal to 12 INCH
[       OK ] LengthTest::1 FEET should equal to 12 INCH(13 us)
[----------] 1 tests from LengthTest

[----------] 1 tests from All Tests

[==========] 1 test cases ran.
[  TOTAL   ] PASS: 1  FAILURE: 0  ERROR: 0  TIME: 13 us
```

Write programs for people first, computers second.

- Steve McConnell

2 用例设计

2.1 Fixture

2.1.1 Fixture

在编写测试时，最费时的部分之一是将整个场景设置成某个已知的状态，并在测试结束后将其复原到初始状态。这个已知的状态称为测试的 Fixture。

示例代码 2-1 test/quantity/LengthTest.cpp

```
#include <magellan/magellan.hpp>

FIXTURE(LengthTest)
{
};
```

FIXTURE 的参数可以是任意的 C/C++ 标识符。一般而言，将其命名为 CUT(Class Under Test) 的名字即可。

2.1.2 独立的 Fixture

使用 Setup/Teardown 分别完成场景的设置，并在测试结束后将其复原到初始状态。其中，SETUP/TEARDOWN 为可选，如果未配置则默认实现为空。

示例代码 2-2 test/quantity/LengthTest.cpp

```
#include <magellan/magellan.hpp>

FIXTURE(LengthTest)
{
    Length length;

    SETUP()
    {}

    TESTDOWN()
    {}

    TEST("length test1")
    {}

    TEST("length test2")
    {}
};
```

执行序列：

1. Length 构造函数
2. SETUP
3. length test1

4. TESTDOWN
5. Length 析构函数
6. Length 构造函数
7. SETUP
8. length test2
9. TESTDOWN
10. Length 析构函数

2.1.3 BDD 风格

xUnit	BDD
FIXTURE	CONTEXT
SETUP	BEFORE
TEARDOWN	AFTER

表 2.1 Magellan 支持的两种 Fixture 风格

2.1.4 共享的 Fixture

假设存在如下的场景，每个 Test 都需要初始化比较耗时的资源，举例如数据库连接、开启浏览器等等。如果一个 Test 自己完成启动和关闭，测试的时间会拖的很长，给持续集成带来了很大的困难。

为此需要在每组不会冲突的 Test 中共享一个浏览器窗口、或数据库的链接，为它们设计共享的 Fixture。

示例代码 2-3 test/quantity/LengthTest.cpp

```
#include <magellan/magellan.hpp>

FIXTURE(LengthTest)
{
    Length length;

    BEFORE_CLASS()
    {}

    AFTER_CLASS()
    {}

    BEFORE()
    {}

    AFTER()
    {}

    TEST("length test1")
    {}

    TEST("length test2")
    {}
};
```

test1 和 test2 的执行序列：

1. BEFORE_CLASS
2. Length 构造函数

3. BEFORE
4. length test1
5. AFTER
6. Length 析构函数
7. Length 构造函数
8. BEFORE
9. length test2
10. AFTER
11. Length 析构函数
12. AFTER_CLASS

2.1.5 全局的 Fixture

有时候需要在所有用例启动之前完成一次性的全局性的配置，在所有用例运行完成之后完成一次性的清理工作。Magellan 则使用 BeforeAll 和 AfterAll 两个关键字来支持这样的特性的。

示例代码 2-4 test/DatabaseTest.cpp

```
#include <magellan/magellan.hpp>

BEFORE_ALL("before all 1")
{
}

BEFORE_ALL("before all 2")
{
}

AFTER_ALL("after all 1")
{
}

AFTER_ALL("after all 2")
{
}
```

BeforeAll 和 AfterAll 向系统注册 Hook 即可，Magellan 便能自动地发现它们，并执行它们。犹如 C/C++ 不能保证各源文件中全局变量初始化的顺序一样，避免在源文件之间的 BeforeAll 设计不合理的依赖关系。

示例代码 2-5 test/quantity/LengthTest.cpp

```
#include <magellan/magellan.hpp>

FIXTURE(LengthTest)
{
    Length length;

    BEFORE_CLASS()
    {}

    AFTER_CLASS()
    {}

    BEFORE()
    {}

    AFTER()
    {}

    TEST("length test1")
    {}

    TEST("length test2")
    {}
};
```

示例代码 2-6 test/quantity/VolumeTest.cpp

```
#include <magellan/magellan.hpp>

FIXTURE(VolumeTest)
{
    Volume volume;

    BEFORE_CLASS()
    {}

    AFTER_CLASS()
    {}

    BEFORE()
    {}

    AFTER()
    {}

    TEST("volume test1")
    {}

    TEST("volume test1")
    {}
};
```

Magellan 可能的执行序列：

1. before all 1
2. before all 2
3. LengthTest::BEFORE_CLASS
4. Length 构造函数
5. LengthTest::BEFORE
6. length test1
7. LengthTest::AFTER
8. Length 析构函数
9. Length 构造函数
10. LengthTest::BEFORE
11. length test2
12. LengthTest::AFTER
13. Length 析构函数
14. LengthTest::AFTER_CLASS
15. VolumeTest::BEFORE_CLASS
16. Volume 构造函数
17. LengthTest::BEFORE
18. volume test1
19. LengthTest::AFTER
20. Volume 析构函数
21. Volume 构造函数
22. LengthTest::BEFORE
23. volume test2
24. LengthTest::AFTER
25. Volume 析构函数

26. VolumeTest::AFTER_CLASS
27. after all 2
28. after all 1

2.2 Test

2.2.1 自动标识

Magellan 能够自动地实现测试用例的标识功能，用户可以使用字符串来解释说明测试用例的意图，使得用户在描述用例时更加自然和方便。

示例代码 2-7 test/quantity/LengthTest.cpp

```
#include <magellan/magellan.hpp>
#include "quantity/length/Length.h"

USING_HAMCREST_NS

FIXTURE(LengthTest)
{
    TEST("1 FEET should equal to 12 INCH")
    {
        ASSERT_THAT(Length(1, FEET), eq(Length(12, INCH)));
    }

    TEST("1 YARD should equal to 3 FEET")
    {
        ASSERT_THAT(Length(1, YARD), eq(Length(3, FEET)));
    }

    TEST("1 MILE should equal to 1760 YARD")
    {
        ASSERT_THAT(Length(1, MILE), eq(Length(1760, YARD)));
    }
};
```

2.2.2 面向对象

Magellan 实现 xUnit 时非常巧妙，使得用户设计用例时更加面向对象。

示例代码 2-8 test/robot-cleaner/RobotCleanerTest.cpp

```
#include "magellan/magellan.hpp"
#include "robot-cleaner/RobotCleaner.h"
#include "robot-cleaner/Position.h"
#include "robot-cleaner/Instructions.h"

USING_HAMCREST_NS

FIXTURE(RobotCleanerTest)
{
    TEST("at the beginning, the robot should be in at the initial position")
    {
        RobotCleaner robot;

        ASSERT_THAT(robot.getPosition(), is(Position(0, 0, NORTH)));
    }

    TEST("left instruction: 1-times")
    {
        RobotCleaner robot;

        robot.exec(left());

        ASSERT_THAT(robot.getPosition(), is(Position(0, 0, WEST)));
    }

    TEST("left instruction: 2-times")
    {
        RobotCleaner robot;

        robot.exec(left());
        robot.exec(left());

        ASSERT_THAT(robot.getPosition(), is(Position(0, 0, SOUTH)));
    }
};
```

消除重复

用例之间存在重复代码，为了改善设计，首先将所有用例使用的 RobotCleaner 对象直接定义在类体里即可。但在运行时，每个用例可得到独立的 RobotCleaner 实例。

示例代码 2-9 test/robot-cleaner/RobotCleanerTest.cpp

```
#include "magellan/magellan.hpp"
#include "robot-cleaner/RobotCleaner.h"
#include "robot-cleaner/Position.h"
#include "robot-cleaner/Instructions.h"

USING_HAMCREST_NS

FIXTURE(RobotCleanerTest)
{
    RobotCleaner robot;

    TEST("at the beginning, the robot should be in at the initial position")
    {
        ASSERT_THAT(robot.getPosition(), is(Position(0, 0, NORTH)));
    }

    TEST("left instruction: 1-times")
    {
        robot.exec(left());
        ASSERT_THAT(robot.getPosition(), is(Position(0, 0, WEST)));
    }

    TEST("left instruction: 2-times")
    {
        robot.exec(left());
        robot.exec(left());
        ASSERT_THAT(robot.getPosition(), is(Position(0, 0, SOUTH)));
    }
};
```

函数提取

提取的相关子函数，可以直接放在 Fixture 的内部，使得用例与其的距离最近，更加体现类作用域的概念。

示例代码 2-10 test/robot-cleaner/RobotCleanerTest.cpp

```
#include "magellan/magellan.hpp"
#include "robot-cleaner/RobotCleaner.h"
#include "robot-cleaner/Position.h"
#include "robot-cleaner/Instructions.h"

USING_HAMCREST_NS

FIXTURE(RobotCleanerTest)
{
    RobotCleaner robot;

    void WHEN_I_send_instruction(Instruction* instruction)
    {
        robot.exec(instruction);
    }

    void AND_I_send_instruction(Instruction* instruction)
    {
        WHEN_I_send_instruction(instruction);
    }

    void THEN_the_robot_cleaner_should_be_in(const Position& position)
    {
        ASSERT_THAT(robot.getPosition(), is(position));
    }

    TEST("at the beginning")
    {
        THEN_the_robot_cleaner_should_be_in(Position(0, 0, NORTH));
    }

    TEST("left instruction: 1-times")
    {
        WHEN_I_send_instruction(left());
        THEN_the_robot_cleaner_should_be_in(Position(0, 0, WEST));
    }

    TEST("left instruction: 2-times")
    {
        WHEN_I_send_instruction(repeat(left(), 2));
        THEN_the_robot_cleaner_should_be_in(Position(0, 0, SOUTH));
    }

    TEST("left instruction: 3-times")
```

```
    {
        WHEN_I_send_instruction(repeat(left(), 3));
        THEN_the_robot_cleaner_should_be_in(Position(0, 0, EAST));
    }

    TEST("left instruction: 4-times")
    {
        WHEN_I_send_instruction(repeat(left(), 4));
        THEN_the_robot_cleaner_should_be_in(Position(0, 0, NORTH));
    }
};
```

This page is intentionally left blank.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand.

- Martin Flower

3 断言

3.1 ASSERT_THAT

Magellan 只支持一种断言原语：ASSERT_THAT，从而避免用户在 ASSERT_EQ/ASSERT_NE, ASSERT_TRUE/ASSERT_FALSE 之间做选择时的困扰，使其断言更加具有统一性。此外，ASSERT_THAT 使得断言更加具有表达力，更加符合语言习惯。

示例代码 3-1 test/hamcrest/CloseToTest.cpp

```
#include <magellan/magellan.hpp>

FIXTURE(CloseToTest)
{
    TEST("double")
    {
        ASSERT_THAT(1.0, close_to(1.0, 0.5));
        ASSERT_THAT(0.5, close_to(1.0, 0.5));
        ASSERT_THAT(1.5, close_to(1.0, 0.5));
    }
};
```

3.2 EXPECT

xUnit	BDD
ASSERT_THAT	EXPECT

表 3.1 Magellan 支持的两种断言关键字

3.3 Matcher

3.3.1 Anything

匹配器	说明
anything	总是匹配
_	anything 语法糖

表 3.2 anything

示例代码 3-2 test/hamcrest/AnythingTest.cpp

```
#include <magellan/magellan.hpp>

USING_HAMCREST_NS

FIXTURE(AnythingTest)
{
    TEST("should always be matched")
    {
        ASSERT_THAT(1, anything<int>());
```

```
        ASSERT_THAT(1u, anything<unsigned int>());
        ASSERT_THAT(1.0, anything<double>());
        ASSERT_THAT(1.0f, anything<float>());
        ASSERT_THAT(false, anything<bool>());
        ASSERT_THAT(true, anything<bool>());
        ASSERT_THAT(nullptr, anything<std::nullptr_t>());
    }

    TEST("should support _ as syntactic sugar")
    {
        ASSERT_THAT(1u, _(int));
        ASSERT_THAT(1.0f, _(float));
        ASSERT_THAT(false, _(int));
        ASSERT_THAT(nullptr, _(std::nullptr_t));
    }
};
```

3.3.2 比较器

匹配器	说明
eq	相等
ne	不相等
lt	小于
gt	大于
le	小于或等于
ge	大于或等于

表 3.3 比较的 Matcher

示例代码 3-3 test/hamcrest/ComparableTest.cpp

```
#include <magellan/magellan.hpp>

USING_HAMCREST_NS

FIXTURE(EqualToTest)
{
    TEST("should allow compare to integer")
    {
        ASSERT_THAT(0xFF, eq(0xFF));
        ASSERT_THAT(0xFF, is(eq(0xFF)));

        ASSERT_THAT(0xFF, is(0xFF));
        ASSERT_THAT(0xFF == 0xFF, is(true));
    }

    TEST("should allow compare to bool")
    {
        ASSERT_THAT(true, eq(true));
        ASSERT_THAT(false, eq(false));
    }

    TEST("should allow compare to string")
    {
        ASSERT_THAT("hello", eq("hello"));
        ASSERT_THAT("hello", eq(std::string("hello")));
        ASSERT_THAT(std::string("hello"), eq(std::string("hello")));
    }
};

FIXTURE(NotEqualToTest)
{
    TEST("should allow compare to integer")
    {
        ASSERT_THAT(0xFF, ne(0xEE));

        ASSERT_THAT(0xFF, is_not(0xEE));
        ASSERT_THAT(0xFF, is_not(eq(0xEE)));
        ASSERT_THAT(0xFF != 0xEE, is(true));
    }

    TEST("should allow compare to boolean")
    {
        ASSERT_THAT(true, ne(false));
        ASSERT_THAT(false, ne(true));
    }

    TEST("should allow compare to string")
    {
        ASSERT_THAT("hello", ne("world"));
        ASSERT_THAT("hello", ne(std::string("world")));
        ASSERT_THAT(std::string("hello"), ne(std::string("world")));
    }
};
```

匹配器	说明
is	可读性装饰器
is_not	可读性装饰器

表 3.4 修饰的 Matcher

3.3.3 修饰器

示例代码 3-4 test/hamcrest/IsNotTest.cpp

```
#include <magellan/magellan.hpp>

USING_HAMCREST_NS

FIXTURE(IsNotTest)
{
    TEST("integer")
    {
        ASSERT_THAT(0xFF, is_not(0xEE));
        ASSERT_THAT(0xFF, is_not(eq(0xEE)));
    }

    TEST("string")
    {
        ASSERT_THAT("hello", is_not("world"));
        ASSERT_THAT("hello", is_not(eq("world")));

        ASSERT_THAT("hello", is_not(std::string("world")));
        ASSERT_THAT(std::string("hello"), is_not(std::string("world")));
    }
};
```

3.3.4 空指针

匹配器	说明
nil	空指针

表 3.5 空指针

示例代码 3-5 test/hamcrest/NilTest.cpp

```
#include <magellan/magellan.hpp>

USING_HAMCREST_NS

FIXTURE(NilTest)
{
    TEST("equal_to")
    {
        ASSERT_THAT(nullptr, eq(nullptr));
        ASSERT_THAT(0, eq(NULL));
        ASSERT_THAT(NULL, eq(NULL));
        ASSERT_THAT(NULL, eq(0));
    }

    TEST("is")
    {
        ASSERT_THAT(nullptr, is(nullptr));
        ASSERT_THAT(nullptr, is(eq(nullptr)));

        ASSERT_THAT(0, is(0));
        ASSERT_THAT(NULL, is(NULL));
        ASSERT_THAT(0, is(NULL));
        ASSERT_THAT(NULL, is(0));
    }

    TEST("nil")
    {
        ASSERT_THAT((void*)NULL, nil());
        ASSERT_THAT((void*)0, nil());
        ASSERT_THAT(nullptr, nil());
    }
};
```

匹配器	说明
contains_string	断言是否包含子串
contains_string_ignoring_case	忽略大小写，断言是否包含子串
starts_with	断言是否以该子串开头
starts_with_ignoring_case	忽略大小写，断言是否以该子串开头
ends_with	断言是否以该子串结尾
ends_with_ignoring_case	忽略大小写，断言是否以该子串结尾

表 3.6 字符串的 Matcher

3.3.5 字符串

示例代码 3-6 test/hamcrest/StartsWithTest.cpp

```
#include <magellan/magellan.hpp>

USING_HAMCREST_NS

FIXTURE(StartsWithTest)
{
    TEST("case sensitive")
    {
        ASSERT_THAT("ruby-cpp", starts_with("ruby"));
        ASSERT_THAT("ruby-cpp", is(starts_with("ruby")));

        ASSERT_THAT(std::string("ruby-cpp"), starts_with("ruby"));
        ASSERT_THAT("ruby-cpp", starts_with(std::string("ruby")));
        ASSERT_THAT(std::string("ruby-cpp"), starts_with(std::string("ruby")));
    }

    TEST("ignoring case")
    {
        ASSERT_THAT("ruby-cpp", starts_with_ignoring_case("Ruby"));
        ASSERT_THAT("ruby-cpp", is(starts_with_ignoring_case("Ruby")));

        ASSERT_THAT(std::string("ruby-cpp"), starts_with_ignoring_case("RUBY"));
        ASSERT_THAT("Ruby-Cpp", starts_with_ignoring_case(std::string("rUBY")));
        ASSERT_THAT(std::string("RUBY-CPP"), starts_with_ignoring_case(std::string("ruby")));
    }
};
```

3.3.6 浮点数

匹配器	说明
close_to	断言浮点数近似等于
nan	断言浮点数不是一个数字

表 3.7 浮点数的 Matcher

示例代码 3-7 test/hamcrest/NanTest.cpp

```
#include <magellan/magellan.hpp>
#include <math.h>

USING_HAMCREST_NS

FIXTURE(IsNanTest)
{
    TEST("double")
    {
        ASSERT_THAT(sqrt(-1.0), nan());
        ASSERT_THAT(sqrt(-1.0), is(nan()));

        ASSERT_THAT(1.0/0.0,  is_not(nan()));
        ASSERT_THAT(-1.0/0.0, is_not(nan()));
    }
};
```

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

- C.A.R. Hoare

4 程序选项

4.1 Option

4.1.1 选项列表

选项	可选值 (* 为默认值)	说明
- -color	[yes*|no]	用于终端打印时颜色的控制
- -format	[stdout*|xml|progress]	用于终端打印输出的格式控制，其中 progress 用于打印进度条
- -filter	regex	匹配执行特定规则的用例集，其中默认执行所有用例
- -break_on_failure	[yes|no*]	当执行失败时，是否立即停止运行
- -list_tests	[yes|no*]	不执行用例，仅罗列出用例
- -repeat	[n|-1|1*]	重复地执行用例 n 次，其中 -1 表示无限次

表 4.1 Magellan 程序选项

This page is intentionally left blank.

表格目录

This page is intentionally left blank.