ztest: A Unit Test Framework with GUI

郑辰阳

2025 年 4 月 26 日

需求分析

- 随着复杂系统架构设计能力的培养需求日益凸显，掌握面向对象设计原则与模式化工程实践已成为高级软件工程教育的核心目标。
- 我们团队计划开发一个提供一个灵活、高效且易于使用（带图形用户界面 GUI）的测试工具，旨在提供一个直观、易用的环境，方便开发人员和测试人员编写、运行和管理测试用例。

系统功能

- 提供一个灵活、高效且易于使用（带图形用户界面 GUI）的测试工具。
- 支持测试用例的管理、断言验证、测试执行以及结果报告。
- 支持多种测试类型（如单元测试、集成测试等），并提供详细的测试结果报告。

测试用例管理

- **测试用例的定义**：通过继承 ZtestBase 类或使用模板类 ZtestSingleCase 定义测试用例。
- **测试用例的注册**：使用测试注册中心 ZTestRegistry 动态注册测试用例。
- **测试套件的构建**：通过 ZTestSuite 类组织多个测试用例，支持批量运行。

断言机制

- **EXPECT_EQ**：验证两个值是否相等。
- **ASSERT_TRUE**：验证条件是否为真。
- **异常处理**：如果断言失败，抛出 **ZTestFailureException** 异常。

测试执行器

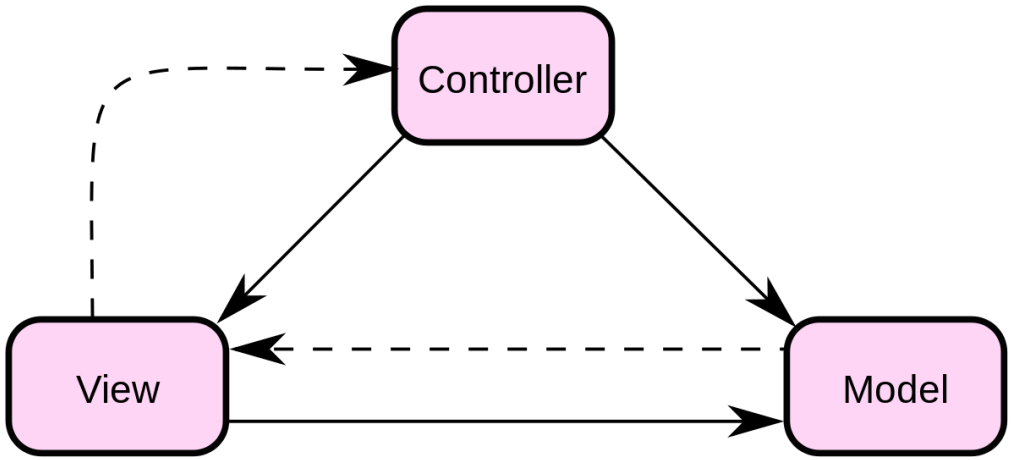
- **ZTestContext**：测试上下文管理类，负责维护测试用例队列，并通过多线程并行执行测试。
- **ZTimer**：用于计时测试的运行时间。
- **测试结果收集**：测试结果存储在 **ZTestResult** 类中。

结果报告

- **ZTestResult**：封装单个测试的结果信息。
- **测试套件报告**：通过 **ZTestSuite** 类生成测试套件的汇总报告。
- **日志输出**：测试结果通过 **ZLogger** 类输出到控制台。

整体架构

- 使用 MVC（Model-View-Controller）模式架构。
- View 关注用户所看见的 UI 并且提供交互。
- Controller 负责控制器作用于模型和视图上。
- Model 主要实现对于底层文件的建模。



技术细节

所有类都以 Z 开头。

- **ZtestInterface**：定义了测试用例的基本接口。
- **ZtestBase**：作为测试用例的基类，实现了测试的基本属性和方法。
- **ZtestSingleCase**：用于定义单个测试用例。
- **ZTestSuite**：用于组织多个测试用例。
- **ZTestRegistry**：测试注册中心，负责注册和管理测试用例。
- **ZTestContext**：测试上下文管理类。

链式调用

工厂模式和构建器模式结合使用，实现链式调用。

- **TestFactory**：工厂类，帮助定义测试用例。
- **TestBuilder**：构建器类，用于构建测试用例。

```
auto test_case = TestFactory::createTest("Addition", ZType
    ::ZSAFE, "", add, 2, 3)
                 .setExpectedOutput(5)
                 .beforeAll([]() { logger.info("
                     Initializing␣test␣case␣resources...\n
                     "); })
                 .afterEach([]() { logger.info("Cleaning␣
                     up␣after␣test...\n"); })
                 .build();
```

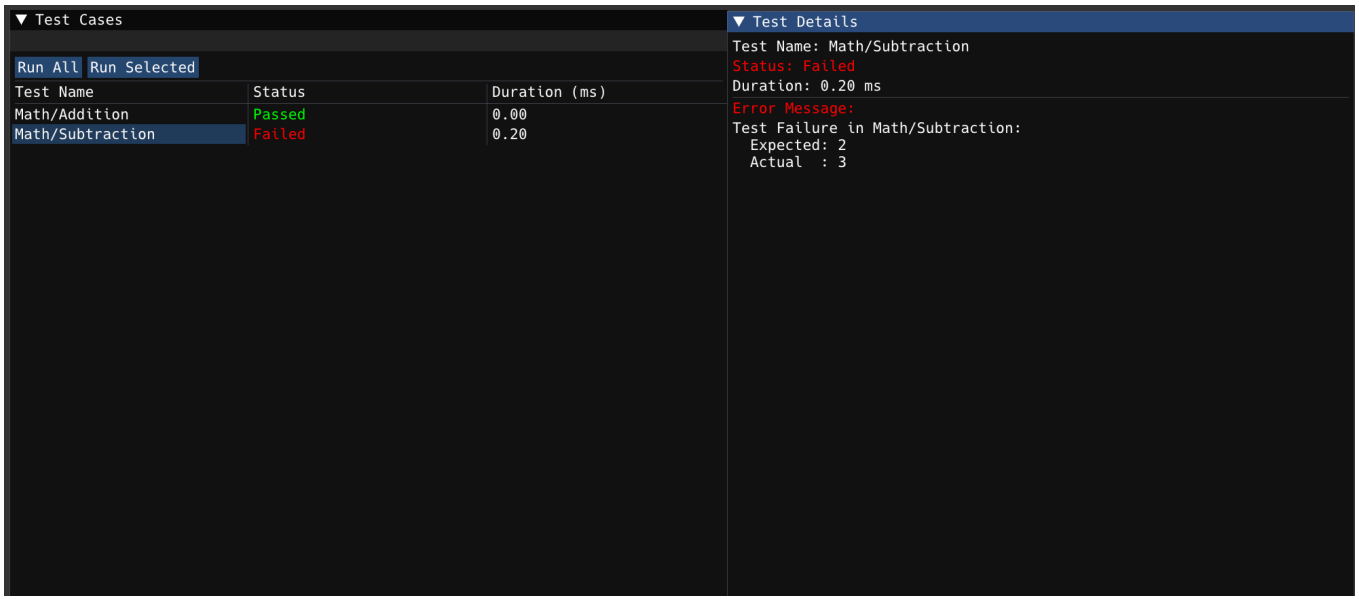
类 gtest 的调用方式

使用宏展开，实现 ASSERT_TRUE 和 EXPECT_EQ。

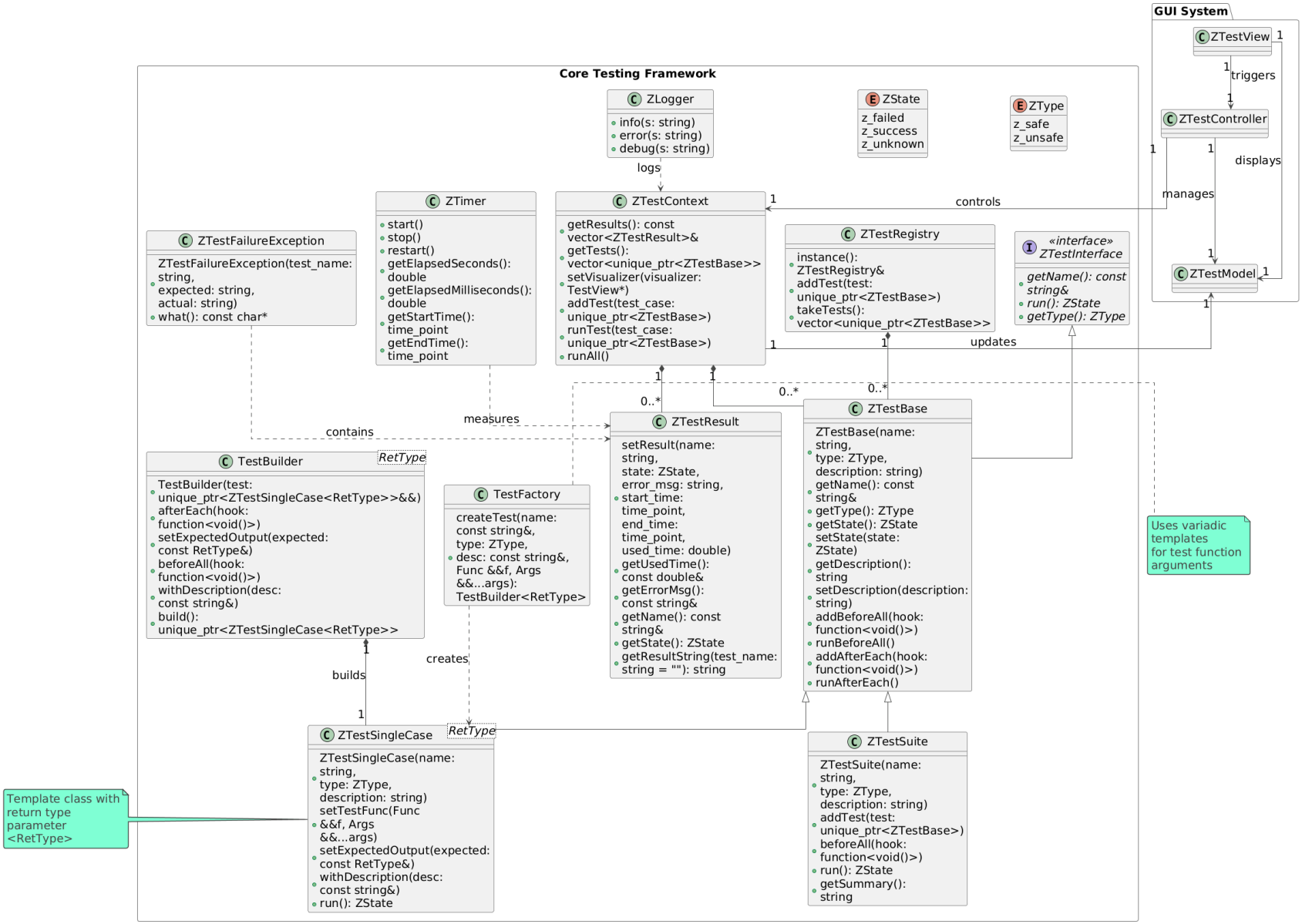
```
TEST(MathTests, Addition) {
    EXPECT_EQ(2 + 2, 4);
}
```

UI 实现

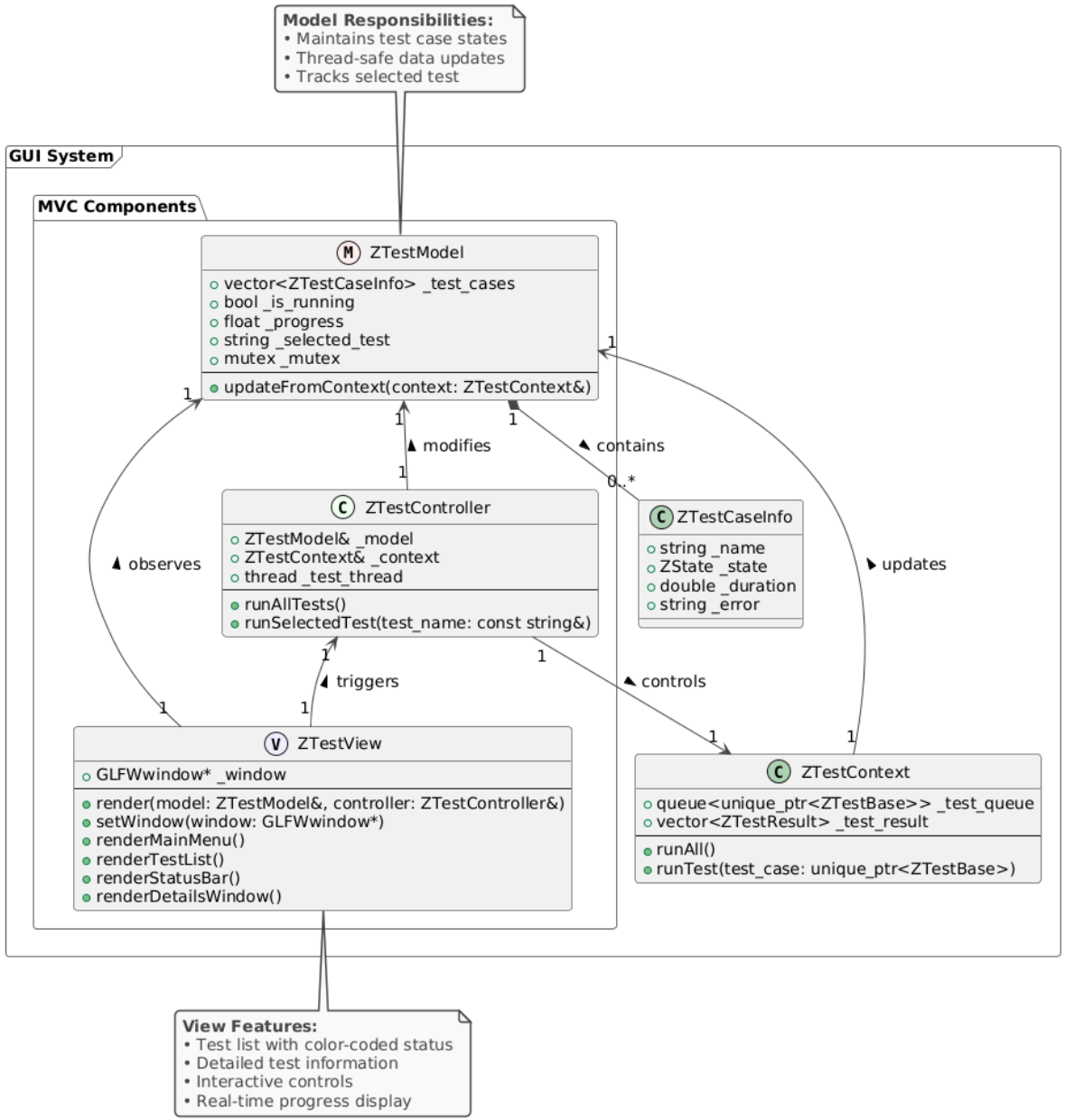
- 使用 MVC 架构实现图形用户界面。
- 使用 Dear ImGui 框架构建可视化测试管理界面。
- 包括模型管理、视图渲染和控制器交互三部分。



UML 类图


GUI System
ZTestView
triggers
ZTestController
displays
manages
ZTestModel
Core Testing Framework
ZLogger
info(s: string)
error(s: string)
debug(s: string)
logs
ZState
z_failed
z_success
z_unknown
ZType
z_safe
z_unsafe
controls
updates
ZTimer
start()
stop()
restart()
getElapsedSeconds(): double
getElapsedMilliseconds(): double
getStartTime(): time_point
getEndTime(): time_point
ZTestContext
getResults(): const vector<ZTestResult>&
getTests(): vector<unique_ptr<ZTestBase>>
setVisualizer(visualizer: TestView*)
addTest(test_case: unique_ptr<ZTestBase>)
runTest(test_case: unique_ptr<ZTestBase>)
runAll()
ZTestRegistry
instance(): ZTestRegistry&
addTest(test: unique_ptr<ZTestBase>)
takeTests(): vector<unique_ptr<ZTestBase>>
«interface» ZTestInterface
getName(): const string&
run(): ZState
getType(): ZType
ZTestFailureException
ZTestFailureException(test_name: string, expected: string, actual: string)
what(): const char*
measures
contains
ZTestResult
setResult(name: string, state: ZState, error_msg: string, start_time: time_point, end_time: time_point, used_time: double)
getUsedTime(): const double&
getErrorMsg(): const string&
getName(): const string&
getState(): ZState
getResultString(test_name: string = ""): string
ZTestBase
ZTestBase(name: string, type: ZType, description: string)
getName(): const string&
getType(): ZType
getState(): ZState
setState(state: ZState)
getDescription(): string
setDescription(description: string)
addBeforeAll(hook: function<void()>)
runBeforeAll()
addAfterEach(hook: function<void()>)
runAfterEach()
Uses variadic templates for test function arguments
TestBuilder
RetType
TestBuilder(test: unique_ptr<ZTestSingleCase<RetType>>&&)
afterEach(hook: function<void()>)
setExpectedOutput(expected: const RetType&)
beforeAll(hook: function<void()>)
withDescription(desc: const string&)
build(): unique_ptr<ZTestSingleCase<RetType>>
TestFactory
createTest(name: const string&, type: ZType, desc: const string&, Func &&f, Args &&...args): TestBuilder<RetType>
builds
creates
ZTestSingleCase
RetType
ZTestSingleCase(name: string, type: ZType, description: string)
setTestFunc(Func &&f, Args &&...args)
setExpectedOutput(expected: const RetType&)
withDescription(desc: const string&)
run(): ZState
ZTestSuite
ZTestSuite(name: string, type: ZType, description: string)
addTest(test: unique_ptr<ZTestBase>)
beforeAll(hook: function<void()>)
run(): ZState
getSummary(): string
Template class with return type parameter <RetType>

MVC 类图


Model Responsibilities:
• Maintains test case states
• Thread-safe data updates
• Tracks selected test
GUI System
MVC Components
ZTestModel
vector<ZTestCaseInfo> _test_cases
bool _is_running
float _progress
string _selected_test
mutex _mutex
updateFromContext(context: ZTestContext&)
modifies
contains
ZTestController
ZTestModel& _model
ZTestContext& _context
thread _test_thread
runAllTests()
runSelectedTest(test_name: const string&)
ZTestCaseInfo
string _name
ZState _state
double _duration
string _error
observes
updates
triggers
controls
ZTestView
GLFWwindow* _window
render(model: ZTestModel&, controller: ZTestController&)
setWindow(window: GLFWwindow*)
renderMainMenu()
renderTestList()
renderStatusBar()
renderDetailsWindow()
ZTestContext
queue<unique_ptr<ZTestBase>> _test_queue
vector<ZTestResult> _test_result
runAll()
runTest(test_case: unique_ptr<ZTestBase>)
View Features:
• Test list with color-coded status
• Detailed test information
• Interactive controls
• Real-time progress display

一些例子

- 定义单个测试用例。
- 使用测试工厂创建测试用例。
- 构建测试套件。
- 运行测试并输出结果。
- 使用断言。
- 测试结果的输出。
- 动态注册测试用例。

定义单个测试用例

```
class MathTests_Addition : public ZtestBase {
public:
    MathTests_Addition()
        : ZtestBase("MathTests.Addition", ZType::ZSAFE, "
            Test␣addition␣function") {}

    Zstate run() override {
        EXPECT_EQ(5, add(2, 3)); // 预期结果为5
        EXPECT_EQ(0, add(0, 0)); // 预期结果为0
        return Zstate::Z_SUCCESS;
    }
};
```

使用测试工厂创建测试用例

```
auto test_case = TestFactory::createTest("Addition", ZType
    ::ZSAFE, "", add, 2, 3)
                    .setExpectedOutput(5)
                    .beforeAll([]() { logger.info("
                        Initializing␣test␣case␣resources...\
                        n"); })
                    .afterEach([]() { logger.info("Cleaning␣
                        up␣after␣test...\n"); })
                    .build();
```

构建测试套件

```
auto suite = std::make_unique<ZTestSuite>("MathTests",
    ZType::ZSAFE, "Mathematical␣operations␣tests");
suite->addTest(TestFactory::createTest("Addition", ZType::
    ZSAFE, "", add, 2, 3)
              .setExpectedOutput(5)
              .build());
suite->addTest(TestFactory::createTest("NegativeNumbers",
    ZType::ZSAFE, "", add, -2, -3)
              .setExpectedOutput(-5)
              .build());
```

运行测试并输出结果

```
ZTestContext context;
context.add_test(std::move(suite)); // 添加测试套件
context.run_all(); // 运行所有测试
```

使用断言

```
ZTEST_F(MathTests, Addition) {
    EXPECT_EQ(5, add(2, 3)); // 预期结果为5
    EXPECT_EQ(0, add(0, 0)); // 预期结果为0
    return Zstate::Z_SUCCESS;
}
```

测试结果的输出

```
ZTestResult result;
result.set_result(Zstate::Z_SUCCESS, "", start_time,
    end_time, elapsed_time);
logger.info(result.get_result_string("MathTests.Addition")
     + "\n");
```

动态注册测试用例

```
ZTEST_F(MathTests, NegativeNumbers) {
    ASSERT_TRUE(add(-2, -3) == -5); // 断言结果为-5
    return Zstate::Z_SUCCESS;
}
```