单元测试：单元测试执行：Mock 对象和 Stub 对象的使用

1 单元测试基础

1.1 单元测试的概念

单元测试是软件开发过程中的一个重要组成部分，它是一种测试方法，旨在对软件中的最小可测试单元进行检查和验证。这些单元通常是函数、方法或类。单元测试的目的是确保每个单元在独立运行时能够正确地执行其预期功能，从而为构建更复杂的应用程序提供坚实的基础。

1.1.1 例子

假设我们有一个简单的函数，用于计算两个数字的和：

```
# 文件名: calculator.py
def add(x, y):
    """返回两个数字的和"""
    return x + y
```

我们可以为这个函数编写一个单元测试，使用 Python 的 unittest 框架：

```
# 文件名: test_calculator.py
import unittest
from calculator import add

class TestAdd(unittest.TestCase):
    def test_add(self):
        """测试 add 函数的正确性"""
        self.assertEqual(add(1, 2), 3)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)

if __name__ == '__main__':
    unittest.main()
```

1.2 单元测试的重要性

单元测试的重要性在于它能够帮助开发者在开发过程中及时发现和修复错误，减少后期集成和系统测试时的问题。它还能作为代码的文档，清晰地展示函数的预期行为。此外，单元测试支持重构，因为它们可以确保在代码修改后，功能仍然按预期工作。

1.3 单元测试的生命周期

单元测试的生命周期通常包括以下几个阶段：

1. **编写测试**：在编写代码之前或之后，编写测试用例来描述代码的预期行为。
2. **运行测试**：执行测试用例，检查代码是否按预期工作。
3. **调试失败的测试**：如果测试失败，定位问题并修复代码。
4. **重构代码**：在确保测试通过后，可以安全地重构代码，因为测试会验证重构后的代码是否仍然正确。
5. **持续集成**：将单元测试集成到持续集成流程中，确保每次代码提交后都能自动运行测试。

1.3.1 例子

假设我们有一个类 User，它有一个方法 login，用于模拟用户登录：

```python
# 文件名: user.py
class User:
    def __init__(self, username, password):
        self.username = username
        self.password = password
        self.is_logged_in = False

    def login(self):
        """模拟用户登录过程"""
        if self.username == "admin" and self.password == "123456":
            self.is_logged_in = True
        else:
            self.is_logged_in = False
```

我们可以为 login 方法编写单元测试，并在代码修改后重新运行测试：

```python
# 文件名: test_user.py
import unittest
from user import User

class TestUser(unittest.TestCase):
    def setUp(self):
        """在每个测试方法前创建一个 User 实例"""
        self.user = User("admin", "123456")

    def test_login(self):
        """测试 login 方法是否正确设置登录状态"""
        self.user.login()
        self.assertTrue(self.user.is_logged_in)
```