单元测试：单元测试执行：高级单元测试技巧：Mockito

1 单元测试基础

1.1 理解单元测试的重要性

单元测试是软件开发过程中的一个重要组成部分，它通过测试软件中的最小可测试单元，如函数或方法，来确保代码的正确性和稳定性。单元测试的重要性在于：

- **提高代码质量**：通过编写单元测试，开发者可以确保代码在修改或重构后仍然能够按预期工作。
- **快速定位问题**：当测试失败时，单元测试可以帮助快速定位到代码中的具体问题，减少调试时间。
- **促进代码重构**：单元测试为代码重构提供了安全网，开发者可以放心地进行代码优化，只要测试仍然通过，代码的功能就没有改变。
- **文档作用**：良好的单元测试可以作为代码的活文档，帮助新加入的开发者理解代码的逻辑和功能。

1.2 JUnit 简介与使用

JUnit 是一个用于 Java 编程语言的单元测试框架，它简化了测试的编写和执行过程。JUnit 的核心概念包括：

- **测试用例**：一个测试用例通常对应一个类，包含多个测试方法。
- **测试方法**：测试方法用于验证类中的某个方法或一组方法的正确性。
- **断言**：用于检查测试结果是否符合预期，如 assertEquals 用于比较两个值是否相等。

1.2.1 JUnit 的使用示例

下面是一个使用 JUnit 进行单元测试的简单示例：

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    private Calculator calculator = new Calculator();

    /**
     * 测试加法功能
     */
```

```
    @Test
    public void testAdd() {
        int result = calculator.add(5, 3);
        assertEquals(8, result);
    }

    /**
     * 测试减法功能
     */
    @Test
    public void testSubtract() {
        int result = calculator.subtract(5, 3);
        assertEquals(2, result);
    }
}
```

在这个例子中，CalculatorTest 类包含了两个测试方法，分别用于测试 Calculator 类的加法和减法功能。每个测试方法都使用了 assertEquals 断言来检查方法的返回值是否与预期相符。

1.2.2 JUnit 的高级特性

JUnit 还提供了许多高级特性，如：

- **参数化测试**：允许使用不同的参数集运行同一个测试方法。
- **测试套件**：可以将多个测试类组合在一起，作为一个整体运行。
- **测试监听器**：允许在测试执行的各个阶段执行自定义代码，如设置或清理资源。

1.2.2.1 参数化测试示例

```
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import java.util.Arrays;
import java.util.Collection;

@RunWith(Parameterized.class)
public class CalculatorParameterizedTest {

    private int a;
    private int b;
    private int expected;

    public CalculatorParameterizedTest(int a, int b, int expected) {
```