Teach Me Skills

xUnit, JUnit, TestNG и их использование в автоматизации

Что такое xUnit?

xUnit — это собирательное название семейства фреймворков для модульного тестирования.

Существуют реализации для разных языков программирования:

- Java - JUnit, TestNG
- .Net - NUnit, MSTest
- PHP - PHPUnit
- Python - PyTest
- JS - Mocha

Функциональные возможности

1. Тесты и их запуск (@Test и Test Runners)
2. Конфигурация, Pre-conditions и Post-conditions (@Before, @After)
3. Утверждения (Assertions) (assertTrue, assertEquals)
4. Форматирование результатов тестирования для отчетов

Функциональные возможности

5

Наборы тестов (Groups)

Параметры аннотаций

1. enabled — можно временно отключить, установив значение в false
2. priority - число, указывающее на очередность тестов
3. description - название для отчета
4. alwaysRun - если true, то метод будет выполняться вне зависимости от результатов методов на которые он завязан

Параметры аннотаций

5 invocationCount - количество раз которое нужно повторить тест

6 threadPoolSize - используется вместе с invocationCount, позволяет указать количество параллельных потоков

7 groups — группы к которым принадлежит тест, например “smoke”, “slow”, “user-profile”

8 retryAnalyzer - параметр, указывающий на класс, который будет анализировать результаты теста на предмет перезапуска

Параметризованные тесты

1. **@DataProvider** - метод, который возвращает двумерный массив объектов
2. **name** - опциональный параметр для метода, чтобы указать читабельное имя
3. **dataProvider** - параметр, указывающий на метод с данными (может быть name или метод)
4. Параметры метода можно использовать в тесте

```
@DataProvider(name = "Входящие данные для задачки iTechArt")
public Object[][] inputForITechTask() {
  return new Object[][]{
        {3, "iTech"},
        {5, "Art"},
        {15, "iTechArt"},
        {6, "iTech"},
        {10, "Art"},
        {30, "iTechArt"},
        {1, "Говно, переделывай"},
        {-1, "Говно, переделывай"},
        {1000000000, "Art"},
        {3.3, "Говно, переделывай"}
  };
}

@Test(dataProvider = "Входящие данные для задачки iTechArt")
public void test123(double number, String expectedString) {
  System.out.println(number + " " + expectedString);
}
```

Suite, Group, Test, Class, Method

1. **Suite** - весь пакет тестов. По-сути .xml файл, созданный для TestNG
2. **Group** - группа тестов, конфигурирование через .xml файл, созданный для TestNG
3. **Test** - это НЕ ОБЯЗАТЕЛЬНО один метод или КЛАСС, это все еще может быть какое-то количество классов/методов, конфигурирование через .xml файл, созданный для TestNG
4. **Class** - класс, который содержит в себе тестовые методы. Рекомендуется последним словом всегда использовать *Test.java (например BlaBlaTest.java)
5. **Method** - метод, помеченный аннотацией @Test

Testng xml suite file

Функциональность:

1. Разделение тестов на группы с возможностью запуска отдельно. Допустим: SmokeTest.xml, RegressionTest.xml
2. Возможность запуска в параллели
3. Возможность запуска из командной строки

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Regression Test" parallel="tests" thread-count="2">
  <test name="test 1">
    <groups>
      <run>
        <include name="slow" />
      </run>
    </groups>
    <classes>
      <class name="testng.TestNGExampleTest" />
    </classes>
  </test> <!-- Test -->
  <test name="test 2">
    <classes>
      <class name="testng.TestNGExampleTest" />
    </classes>
  </test> <!-- Test -->
</suite>
```

Считывание параметров извне

1. **@Parameters** - дает возможность считать переменные из .xml файла
2. **@Optional** - подставляет дефолтное значение в случае если данные не будут найдены в xml
3. **<parameter name="" value="">** - передает переменную со значением внутрь тестов

```
@Parameters({"email", "password"})
@Test(groups = {"slow"})
public void test4(@Optional("1")
                  String p1,
                  @Optional("string")
                  String p2) {
  System.out.println(p1 + " " + p2);
}
```

```
  <test name="test 1">
<!--      <groups>
        <run>
          <include name="Писал Юра" />
        </run>
      </groups>-->
      <classes>
        <class name="testng.TestNGExampleTest"/>
        <parameter name="password" value="password"></parameter>
        <parameter name="email"
value="test@mailinator.com"></parameter>
      </classes>
  </test> <!-- Test -->
```

Параллельный запуск

1. Создаем **.xml файл** в формате TestNG (suite -> test -> classes)
2. Указываем **thread-count**
3. Указываем тип распараллеливания **parallel** (методы, классы, тесты)
4. Ниже описываем необходимые тесты/классы/группы

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite thread-count="2" name="Test suite" parallel="classes">
  <test name="test 1">
    <classes>
      <class name="testng.TestNGExampleTest"/>
      <parameter name="password"
value="password"></parameter>
      <parameter name="email"
value="test@mailinator.com"></parameter>
    </classes>
  </test> <!-- Test -->
  <test name="test 2">
    <classes>
      <class name="testng.TestNGExampleTest2"/>
      <class name="testng.TestNGExampleTest3"/>
    </classes>
  </test> <!-- Test -->
</suite>
```

Before and After

1. @BeforeSuite & @AfterSuite
2. @BeforeGroup & @AfterGroup
3. @BeforeTest & @AfterTest
4. @BeforeClass & @AfterClass

Before

5

@BeforeMethod & @AfterMethod

Повторение

1. Имплементируем интерфейс **iRetryAnalyzer**
2. Имплементируем **метод retry** как нам заблагорассудится
3. Используем объект **iTestResult** для получения информации о тесте
4. Используем параметр **retryAnalyzer** для подключения класса к методу

```
public class Retry implements IRetryAnalyzer {

    private int attempt = 1;
    private static final int MAX_RETRY = 3;

    @Override
    public boolean retry(ITestResult iTestResult) {
        if (!iTestResult.isSuccess()) {
            if (attempt < MAX_RETRY) {
                attempt++;
                iTestResult.setStatus(ITestResult.FAILURE);
                System.out.println("Retrying once again");
                return true;
            } else {
                iTestResult.setStatus(ITestResult.FAILURE);
            }
        } else {
            iTestResult.setStatus(ITestResult.SUCCESS);
        }
        return false;
    }

}
```

```
@Test (retryAnalyzer = Retry.class)
public void dependsOn() {
    throw new NullPointerException();
}
```

Действия при падении/успехе

1. Имплементируем интерфейс **ITestListener**
2. Перезаписываем ВСЕ методы, каждый отдельный отвечает за статус теста
3. Над тестовым классом, можно даже над BaseTest указываем аннотацию **@Listeners** и указываем там наш класс