

14 自动化测试框架基础

本章大纲

14.1 自动化框架介绍

14.2 自动化框架的实现

什么是自动化测试框架

UI自动化测试框架是应用于自动化测试的程序框架，它提供了**可重用的自动化测试模块**，提供最基本的自动化测试功能（打开浏览器，输入文字，点击按钮），或提供自动化测试执行和管理功能的架构模块（例如,TestNG）。它是由一个或多个自动化测试基础模块、自动化测试管理模块、自动化测试统计模块等组成的工具集合。

自动化框架常见的模式

- 数据驱动测试框架
- 关键字驱动测试框架
- 混合型测试框架（数据驱动+关键字驱动）
- 行为驱动测试框架

数据驱动测试框架

使用数组、测试数据文件或者数据库等方式作为测试过程输入的自动化测试框架，此框架可以将所有的测试数据在自动化测试执行的过程中进行自动加载，动态判断测试结果是否符合预期，并自动输出测试报告。

关键字驱动测试框架

被操作的元素对象、操作的方法和操作的数据值作为测试过程输入的自动化测试框架。（存在于UFT中）可以保存在数组、文件或数据库中。

例如：输入框，输入，内容

name:username,sendKeys,admin

行为驱动测试框架

行为驱动开发是一种敏捷软件开发技术，Behavior Driven Development。Cucumber是实现BDD开发模式的一种测试框架，实现了自然语言来执行相关联的测试代码的需求。

<https://cucumber.io/>

<http://repo1.maven.org/maven2/info/cukes/>

Cucumber-core.jar 核心包

Cucumber-java.jar 通过 java 编写需要下载这个包

Cucumber-html.jar 生成结果为 html 文件需要下载这个包

Cucumber-testng.jar 使用testng执行测试并生成报告

行为驱动测试框架

eclipse插件下载

<https://cucumber.io/cucumber-eclipse/update-site>

```
Feature: 在百度中搜索

@TestngScenario
  Scenario: 搜索testng
    Given 打开百度, 验证title
    When 输入 "testng"
    Then 点击搜索按钮
    Then 清除搜索框
```


自动化测试框架的作用

- 能够有效组织和管理测试用例
- 进行数据驱动或者关键字驱动测试
- 将基础的代码封装，降低测试脚本编写的复杂性和重复性
- 提高测试脚本维护和修改的效率
- 自动执行测试脚本，并自动发布测试报告，为持续集成的开发方式提供脚本支持
- 让不具备编程能力的测试工程师开展自动化测试工作

自动化测试框架的核心思想

世界上没有万能的自动化测试框架，不要为了自动化而自动化，考虑自动化能解决什么样的问题。

核心思想是将常用的脚本代码或者测试逻辑进行抽象和总结，然后将这些代码进行面向对象设计，将需要复用的代码封装到可公用的类方法中。通过调用公用的类方法，测试类中的脚本复杂度会被大大降低，让更多脚本能力不强的测试人员来实施自动化测试。

自动化测试框架的步骤

1. 根据测试业务的手工测试用例，选出需要自动化的用例（例如：冒烟）
2. 根据可自动化执行的测试用例，分析出测试框架需要模拟的手工操作和重复高的测试流程或逻辑
3. 将重复高的测试流程在代码中实现，并进行封装
4. 根据业务的类型和本身的技术能力选择数据驱动测试、关键字驱动测试框架、混合型框架还是行为测试框架
5. 确定框架类型后，将框架中的常用的浏览器选择、测试数据处理、文件操作、数据库操作、页面元素的原始操作、日志和报告等功能进行方法的封装实现

自动化测试框架的步骤

6. 对框架代码进行集成测试和系统测试，采用PO模式和TestNG框架编写测试脚本，使用框架进行自动化测试，验证框架的功能是否可以满足自动化测试的需求。
7. 编写自动化测试框架的常用API，以供他人参阅
8. 在测试组中内部进行培训和推广
9. 不断收集测试过程中的框架使用问题和反馈意见，不断增加和优化自动化框架的功能，不断增强框架中复杂操作的封装效果，尽量降低测试脚本的编写复杂性
10. 定期评估测试框架的使用效果，评估自动化测试的投入和产出比，再逐步推广自动化框架的应用范围

本章大纲

14.1 自动化框架介绍

14.2 自动化框架的实现

自动化框架的构成

- utils包：实现测试过程中调用的工具类方法，例如：文件操作、mapObject、页面对象的操作方法
- appModules包主要用于实现复用的业务逻辑封装方法
- pageObjects包：用于实现被测试的页面对象
- testCases 包：具体的测试方法
- dataprovider包：提供数据驱动类，txt驱动，excel驱动，数据库的驱动

Properties类

`java.util`

类 `Properties`

`java.lang.Object`

└ `java.util.Dictionary<K, V>`

└ `java.util.Hashtable<Object, Object>`

└ `java.util.Properties`

Properties类


- `getProperty (String key)` , 用指定的键在此属性列表中搜索属性。也就是通过参数 `key` , 得到 `key` 所对应的 `value`。
- `load (InputStream inStream)` , 从输入流中读取属性列表 (键和元素对) 。通过对指定的文件 (比如说上面的 `test.properties` 文件) 进行装载来获取该文件中的所有键 - 值对。以供 `getProperty (String key)` 来搜索。
- `setProperty (String key, String value)` , 调用 `Hashtable` 的方法 `put` 。他通过调用基类的`put`方法来设置 键 - 值对。
- `store (OutputStream out, String comments)` , 以适合使用 `load` 方法加载到 `Properties` 表中的格式 , 将此 `Properties` 表中的属性列表 (键和元素对) 写入输出流。与 `load` 方法相反 , 该方法将键 - 值对写入到指定的文件中去。
- `clear ()` , 清除所有装载的 键 - 值对。该方法在基类中提供。

Properties类

```
public class ReadProperties {  
    public static void main(String[] args) throws IOException {  
        Properties prop= new Properties();  
        FileInputStream in = new FileInputStream("data/object.properties");  
        prop.load(in);  
        in.close();  
        String locator = prop.getProperty("mainpage.login");  
        String locatorValue = new String(locator.getBytes("ISO8859-1"), "UTF-8");  
        System.out.println(locatorValue);  
    }  
}
```

对象库

使用属性文件存储被测试页面上的元素的定位方式和定位表达式，做到定位数据和定位程序的分离。



```
*ObjectMap.java  objectmap.properties  TestObjectMap.java  AddMovie.java

1login.login_link=link:登录
2login.uName=name:username
3login.uPwd=name:password
4login.submitBtn=xpath://input[@value='马上登录']

@Test
public void test1() throws InterruptedException{
    driver.get("http://localhost:8032/MyMovie");
    try {
        driver.findElement(objectmap.getlocator("login.login_link")).click();
        Thread.sleep(5000);
        driver.findElement(objectmap.getlocator("login.uName")).sendKeys("tom01");
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    Thread.sleep(5000);
}
```

封装操作表格的公用类

```
public class Table {

    WebElement table;

    public Table(WebElement table) {
        this.table = table;
    }

    public int getRowCount() {
        List<WebElement> rows = this.table.findElements(By.tagName("tr"));
        return rows.size();
    }

    public int getColCount() {
        List<WebElement> rows = this.table.findElements(By.tagName("tr"));
        return rows.get(0).findElements(By.tagName("td")).size();
    }

    // 获得单元格某行某列的所有内容
    public WebElement getCell(int rowNo, int ColNo) {
        List<WebElement> rows = this.table.findElements(By.tagName("tr"));
        WebElement currentRow = rows.get(rowNo - 1);
        List<WebElement> cols = currentRow.findElements(By.tagName("td"));
        WebElement currentCell = cols.get(ColNo - 1);
        return currentCell;
    }
}
```

数据驱动测试框架的优点

1. 通过配置文件，实现页面元素定位表达式和测试代码的分离
2. 使用ObjectMap方式，简化页面元素定位相关的代码工作量
3. 使用PageObject模式，封装了网页中的页面元素，方便测试代码调用，实现了修改一处全局生效的目标
4. 在appModules包封装了常用的页面对象操作方法简化了脚本编写的工作量
5. 在Excel中定位多个测试数据，测试框架可通过调用测试数据完成数据驱动测试
6. 实现了测试执行过程中的日志记录功能，可以通过日志文件分析测试脚本执行的情况