ЛАНИТ
ЭКСПЕРТИЗА

ПРОГРАММНЫЙ КОМПЛЕКС
BANJO FRAMEWORK

Руководство по установке, настройке и эксплуатации

Москва
2022

Содержание

Аннотация

Настоящий документ содержит описание функциональных характеристик Программного комплекса Banjo Framework (далее – ПО), а также сведения, необходимые для:

- установки ПО;
- конфигурирования ПО;
- эксплуатации ПО;
- поддержания жизненного цикла ПО, в том числе устранения неисправностей и совершенствования ПО;

а также информацию о персонале, необходимом для обеспечения такой поддержки.

Список сокращений

Сокращение	Определение
QA	специалист по обеспечению качества разработки программного обеспечения
АРМ	автоматизированное рабочее место
АТ	автоматизированное тестирование; автотест
ПО, система, фреймворк	Программный комплекс Banjo Framework
СТП	служба технической поддержки
ТК	тест кейс, тестовый сценарий
ФТ	функциональное тестирование; тестовый сценарий для выполнения функционального тестирования

1 Общие сведения

Banjo Framework – коробочное решение для автоматизации тестирования. Продукт предназначен как для тестирования UI, так и для тестирования широкого спектра API. Banjo объединяет управление зависимостями от Selenium WebDriver, Atlas и Citrus Framework, Cucumber и позволяет получать отчетность в Allure Report и Report Portal.

Особенности решения:

- Старт автоматизации в сжатые сроки.
- Библиотека общих шагов позволяет писать простые тестовые сценарии без необходимости написания кода на языке программирования.
- Позволяет писать API-тесты, используя различные протоколы обмена сообщений HTTP REST, JMS, TCP/IP, SOAP, FTP, SSH и другие.
- Для UI тестов есть возможность получения скриншотов в случаях неудачного прохождения сценария, по требованию или после каждого шага теста.
- Использование Citrus в основе решения позволяет получить многофункциональный фреймворк для тестирования API.

2 Описание функциональных характеристик

В основе функциональности Banjo Framework лежат следующие технологические принципы:

1. Интеллектуальная структура автоматизации интуитивно понятна: она может проверять код и создавать автоматизированные тесты, соответствующие изменениям кода.

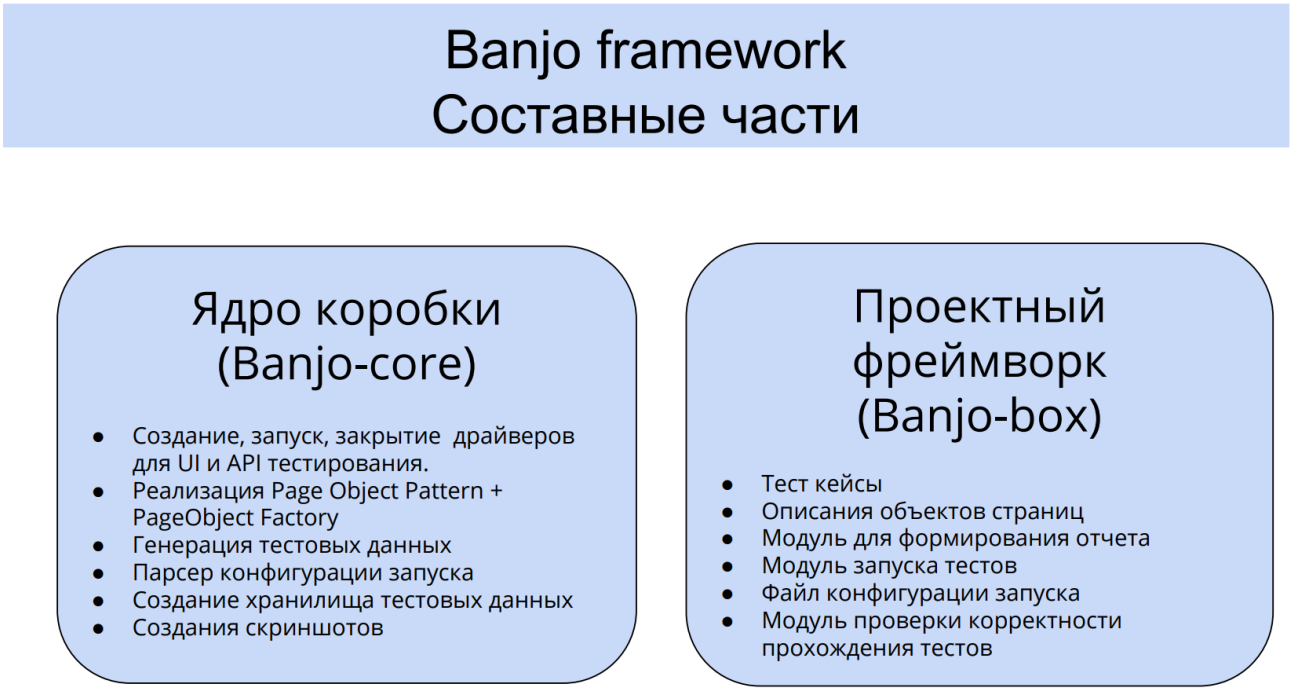
2. Интеллектуальная система автоматизации испытаний динамична: она может использовать методы когнитивных вычислений для динамической идентификации и экранирования элементов, а также для обновления репозиториев объектов.

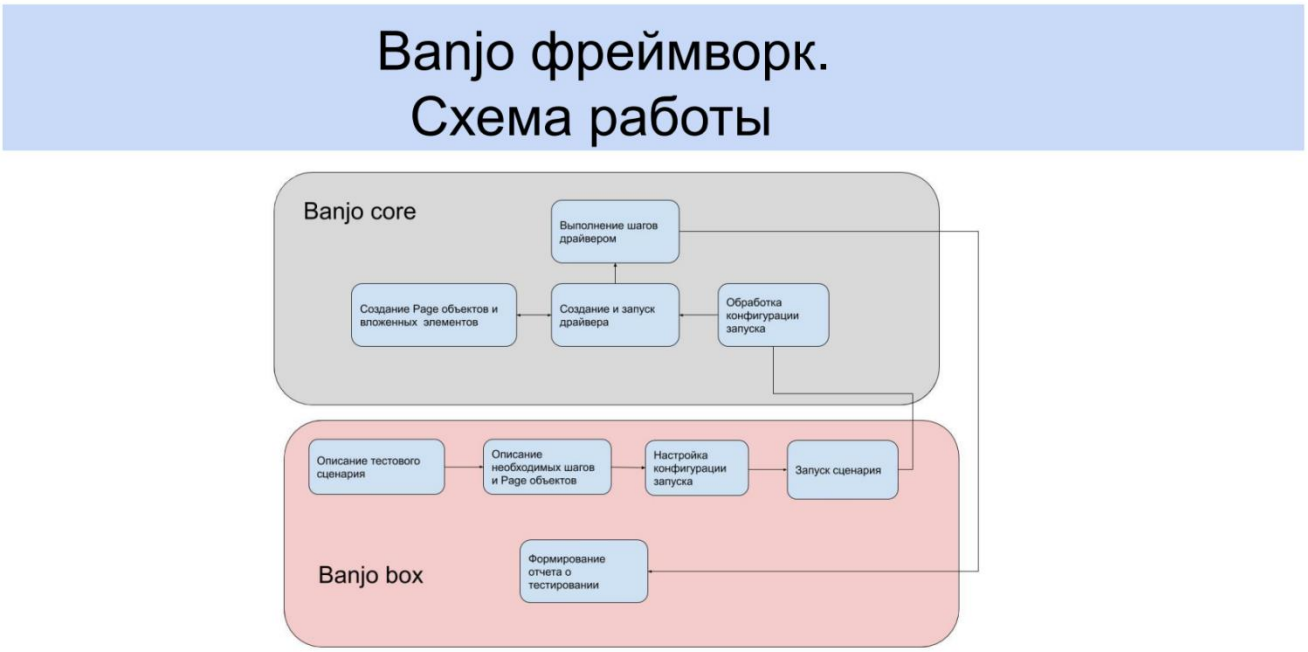
3. Интеллектуальная структура автоматизации испытаний может создавать свою собственную среду: структура может вращать среды во время выполнения через машиночитаемые файлы определений.

4. Интеллектуальная система автоматизации может расставлять приоритеты: фреймворк, который может идентифицировать и выполнять критические тестовые случаи из автоматизированного пакета, чтобы достичь высокого выхода дефектов на каждом тестовом случае с использованием алгоритмов, таких как алгоритм Random Forest.

5. Интеллектуальная система автоматизации предоставляет собственные данные для тестирования: данные могут быть предоставлены либо путем виртуализации, либо путем надстройки, либо путем создания синтетических данных.

6. Возможность применения для всех видов тестирования.





Стек используемых технологий:

- Spring Framework
- Selenium
- Citrus
- Allure
- TestNg
- Atlas framework
- Maven
- Cucumber

2.1 Ядро фреймворка

Ядро фреймворка состоит из следующих основных компонентов:

Менеджер драйверов - служит для организации работы с браузерами. Создаёт экземпляры браузеров с указанными при запуске параметрами. Используется Selenium.

За работу данного менеджера отвечает пакет Driver. Также в том пакете реализован механизм работы с browsermob-proxy, который управляет сервисом прокси.

Менеджер Исключений - обработка ошибок и внештатных ситуаций. Обеспечивает поддержку soft assert (продолжение работы при ошибках низкого приоритета)

За работу данного класса отвечает пакет с одноименным названием Exceptions, который и содержит классы исключений.

Механизм работы с page object – взаимодействие с элементами на странице, в том числе с блочными. Используется библиотека Qameta Atlas.

Блок, отвечающий за имплементацию тестовой логики, состоит из 3 пакетов:

• Make – пакет, содержащий вспомогательные методы для работы с элементами, интеллектуальные ожидания и классы-помощники для работы с javascript.

• Pages – пакет, отвечающий за создание объектов абстрактных страниц, блоков, элементов и взаимодействия между ними.

• Steps – вспомогательные шаги для работы с объектами страниц, блочными элементами и простыми веб-элементами, драйвером и временным тестовым хранилищем.

Менеджер контекста - осуществляет работу в многопоточном окружении. В данном модуле происходит создание и конфигурация бинов ядра. Использует Spring Framework.

За работу с ядром отвечают 2 пакета:

• Context – пакет, отвечающий за конфигурацию бинов ядра и взаимодействия с ними.

• Utils – пакет со вспомогательными классами ядра.

За генерацию тестовых данных отвечает пакет Datageneration, в котором хранятся классы и методы, описывающие и генерирующие различные виды однотипных тестовых данных.

Менеджер интеграции - служит для организации работы с различными протоколами обмена данными, а также организует взаимодействие с различными модулями тестируемого приложения с помощью API. Использует Citrus Framework.

За инициализацию работы фреймворка Citrus Framework отвечает одноименный пакет Citrus.

2.2 Проектный фреймворк

Проектный фреймворк состоит из нескольких компонентов:

Тестовый движок, который предназначен для организации и запуска тестов и сбора информации о результатах тестирования. Использует фреймворк TestNG и Qameta Atlas.

К тестовому движку относятся следующие пакеты:

• Assertion – пакет, содержащий инструменты для реализации проверок тестовых шагов.

• Extensions – пакет, содержащий расширения для Атласа.

• Page – пакет, содержащий описания страниц конкретного проекта, элементы на них, возможные действия с ними.

• Utils – Вспомогательные модули для тестовых шагов, вспомогательные методы и библиотеки, необходимые для расширения и дополнения возможностей других пакетов проектного фреймворка.

• Hooks – операции, выполняемые до и после тестов.

• Steps – реализация Gherkin шагов на языке программирования.

• Api - пакет, включающий в себя набор конфигураций для различных EndPoint, где прописываются, помимо адреса, методы взаимодействия с конкретным EndPoint.

А также каталог с расширениями используемых сторонних библиотек - META.INF.SERVICES

Пакет Config отвечает за конфигурацию и запуска тестов – позволяет настроить параметры запуска автотестов и запустить их выполнение.

Пакет Features – каталог, в котором содержатся feature файлы с тестами, отвечает за тестовую логику фреймворка.

Пакет Resource – каталог с необходимыми ресурсами для Allure

В пакете Templates хранятся сценарии возможных запросов и ответов от приложения, которое тестируется через api.

3 Разворачивание фреймворка

Необходимое ПО для инсталляции фреймворка:

- Open JDK 8 (https://openjdk.java.net/);
- Maven - версия 3.3.1 или выше (https://maven.apache.org/download.cgi).

3.1 Создание пустого проекта и клонирование репозитария

Чтобы запустить Cucumber с Maven:

Создайте новый проект Maven.

Скачайте проект со следующего репозитория - https://github.com/lanit-izh/automation-framework-box.git

Или можно клонировать проект через IDEA следующим образом:

Перейдите в меню VCS | Checkout from Version Control | Git

Затем в диалоговом окне Clone Repository укажите URL-адрес хранилища, которое вы хотите клонировать. URL адрес, который необходимо вести - https://github.com/lanit-izh/automation-framework-box.git

В поле Directory введите путь к папке, в которой будет создан ваш локальный Git-репозиторий.

Нажмите Clone. Если вы хотите создать проект на основе этих источников, нажмите «Да» в диалоговом окне подтверждения. IntelliJ IDEA автоматически установит отображение корня Git в корневой каталог проекта.

3.2 Запуск тестового примера

1. После того как проект был склонирован, можно запустить тестовый пример. Данный пример уже есть в проекте, необходимо его только запустить.

2. Выполните следующую команду: mvn clean test -Dtest.properties=/default.properties,

параметр Dtest.properties указывает, из какого файла необходимо брать настройки для запуска набора тестов. В примере выбран файл с настройками по умолчанию.

3. После выполнения данной команды запустятся все тесты, которые размещены в каталоге features.

4. Результаты выполнения автотестов можно посмотреть в папке allure-results.

3.3 Запуск фреймворка как приложения.

Для запуска проекта введите команду

mvn spring-boot:run

В случае успешного запуска программы по адресу

http://localhost:8090/swagger-ui.html

Вам доступен Swagger.

3.3.1 Дополнительные Параметры

Порт для запускаемого приложения можно отредактировать application.properties файле:

server.port=8090

либо передать при запуске

-Dserver.port=8090

3.3.2 Запуск тестовых feature файлов

Для запуска тестов нужно отправить POST запрос про адресу

http://localhost:8090/test

Передав в качестве параметров:

Feature файл

Json с настройками запуска.

Пример json файла:

{ "site_url": "https:ya.ru", "browser": "chrome", "browser_config": "config/browser.config/chromedriver.config.yaml", "hub_url": "localhost", "remote": "false", "proxy": "false", "proxy_config": "", "alias_input": "", "alias_output": "" }

Полный пример запроса:

curl -X POST "http://localhost:8099/test" -H "accept: text/plain;charset=utf-8" -H "Content-Type: multipart/form-data" -F "feature=@SampleScenarioRU.feature;type=" -F "testProperties={ "site_url": "https:ya.ru", "browser": "chrome", "browser_config": "config/browser.config/chromedriver.config.yaml", "hub_url": "localhost", "remote": "false", "proxy": "false", "proxy_config": "", "alias_input": "", "alias_output": "" }"

Ответом на запрос приходит идентификатор запуска теста - *UUID*.

Зная UUID запущенного теста, с помощью запросов:

- Можно узнать статус запущенного теста.
- http://localhost:8090/status/{uuid}
- Скачать zip-архив, содержащий Allure отчет о прохождении теста.
- http://localhost:8090/allure/download/{uuid}
- Удалить Allure отчет о прохождении теста

- http://localhost:8090/allure/delete/{uuid}

3.3.3 Организация процесса непрерывной интеграции (CI)

1. В файле .github/workflows/docker-image.yml описан пайплайн запуска unit тестов и деплой docker-образа с фреймворком в docker hub.
2. Запуск пайплайна происходит при пуше в ветку, указанную в файле docker-image.yml. Включает в себя 3 джобы: 1.1 Запуск юнит тестов 1.2 После успешного прохождения юнит тестов запускается билд docker-образа с фреймворком 1.3 После успешного билда, запускается деплой docker-образа с фреймворком в docker hub.
3. Для запуска сервера CI необходимо запустить скрипт start-runner. После запуска скрипта поднимется GitHub runner.
4. В файле start-runner прописывается пароль от docker hub для пуша образа. Логин передается в открытом виде в файле docker-image.yml. Также скрипт содержит в себе необходимые параметры для запуска контейнера с раннером. RUNNER_NAME - имя раннера, для отображения на странице https://github.com/<username>/<projectname>/settings/actions GITHUB_ACCESS_TOKEN - токен для доступа к репозиторию github. RUNNER_REPOSITORY_URL - адрес репозитория. RUNNER_WORK_DIRECTORY - рабочая директория раннера. Параметр -v для реализации Docker-in-Docker, alozhkin/github-runner-maven-jdk8 - имя образа с установленным раннером, jdk8, maven, docker. После создания образа происходит создание файла с паролем от docker hub.
5. Отображение статуса раннеров находится на странице https://github.com/<username>/<projectname>/settings/actions.
6. Просмотр выполнения CI на странице https://github.com/<username>/<projectname>/actions

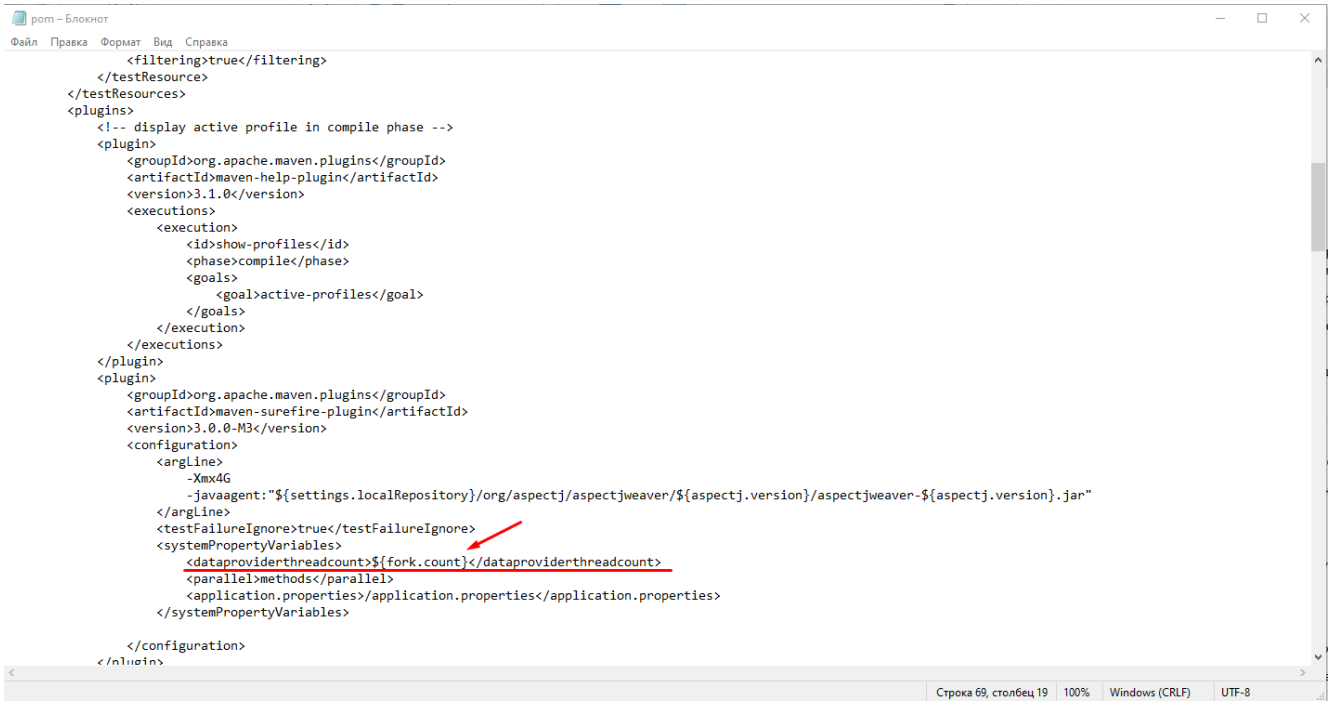
4 Конфигурация и параметры запуска автотестов

4.1 Настройка многопоточности

Необходимое количество потоков для проведения тестирования задается в файле, хранящем описание проекта - **pom.xml**.

За количество потоков отвечает атрибут **${fork.count}**

<dataproviderthreadaccount>${fork.count}</dataproviderthreadaccount>
По умолчанию тесты выполняются в 10 потоков.



4.2 Настройки прокси

Все необходимые настройки для корректной работы через прокси-сервер задаются в файле, расположенном по умолчанию в

…\src\main\resources\config\proxy.config
Параметры, которые необходимо указать, для корректной работы через прокси-сервер:

domainForAutoAuthorization – Строка. Имя домена, на котором должен быть авторизирован прокси.

authUsername – Строка. Имя пользователя для домена, на котором должен быть авторизирован прокси.

authPassword – Строка. Пароль для домена, на котором должен быть авторизирован пользователь.

authType – Тип авторизации. Возможные значения для выбора – BASIC, NTLM.

trustAllServers – Если прокси должен доверять всем серверам, то укажите для данного параметра значение true, иначе false.

Port – номер порта.

StartLocal – Установите true, если прокси должен автоматически запускаться на локальном хосте.

Host – Строка. Хост удаленного прокси.

socksProxy – Строка. SOCKS прокси для удаленного прокси-сервера.

socksUsername – Строка. SOCKS имя пользователя для удаленного прокси-сервера.

socksPassword – Строка. SOCKS пароль для удаленного прокси-сервера.

4.3 Настройки тайм-аутов

При выполнении автотестов часто бывает необходимо воспользоваться тайм-аутами для проверки выполнения того или иного ТК. Данный фреймворк позволяет определить время ожидания на следующие события:

- Выполнение скрипта
- Загрузка страницы
- Появление элемента

Все эти параметры можно указать в

…\src\main\resources\config\timeouts.config

Файл с настройками имеет следующие параметры:

implicitlyWait– время ожидания появления элемента на странице (в секундах)

pageLoadTimeout – время ожидания загрузки страницы (в секундах)

scriptTimeout – время ожидания выполнения скрипта (в секундах)

4.4 Настройка работы с API

Настройка работы с **API** находится по адресу

\src\main\resources\citrus-application
В данном файле на текущий момент есть один параметр,

citrus.spring.java.config=api.EndpointConfig
В нем указывается ссылка на конфигурацию конкретного Endpoint.

Если данный параметр оставить пустым, то все параметры Endpoint необходимо будет указать непосредственно в коде.

Приложение 1. Регламент поддержания жизненного цикла программного обеспечения

Поддержание жизненного цикла ПО обеспечивается за счет его сопровождения и проведения обновлений в соответствии с собственным планом разработки ПО. В рамках технической поддержки ПО оказываются следующие услуги:

- помощь в настройке и администрировании;
- пояснение функционала модулей ПО, помощь в эксплуатации ПО;
- предоставление документации.

1 Устранение неисправностей, выявленных в ходе эксплуатации программного обеспечения

Неисправности, выявленные в ходе эксплуатации ПО, могут быть исправлены следующим образом:

- Массовое автоматическое обновление компонентов ПО;
- Устранение неисправности по запросу пользователя на адрес службы технической поддержки: lanit_exp@lanit.ru.

2 Совершенствование программного обеспечения

ПО регулярно дорабатывается специалистами ООО «ЛАНИТ ЭКСПЕРТИЗА» в соответствии с собственным планом разработки ПО.

Пользователь может самостоятельно повлиять на совершенствование ПО, для этого предложение по усовершенствованию ПО необходимо направить на адрес службы технической поддержки: lanit_exp@lanit.ru

Предложение будет рассмотрено и, в случае признания его эффективности, будет добавлено в план разработки ПО.

3 Техническая поддержка

Для оказания технической поддержки ПО выделен адрес службы технической поддержки: lanit_exp@lanit.ru. По срочным вопросам, связанным с работой ПО, можно позвонить на номер +7 (495) 967-66-50 доб. 16397.

4 Информация о персонале, необходимом для обеспечения поддержки работоспособности «Banjo Framework»

Администраторы «Banjo Framework» должны обладать навыками администрирования операционных систем семейств Linux и навыками работы с:

- Java SE;
- JDK 1.8;
- Maven;
- IntelliJ IDEA.

Для работы администраторы системы должны изучить Руководство администратора «Banjo Framework».

Гарантийное обслуживание программного обеспечения осуществляется силами штатных специалистов службы технической поддержки ООО «ЛАНИТ ЭКСПЕРТИЗА». Команда службы поддержки включает 2 специалистов: Разработчик и Ведущий разработчик.

Техническая поддержка программного обеспечения осуществляется силами штатных специалистов службы технической поддержки ООО «ЛАНИТ ЭКСПЕРТИЗА». Команда службы поддержки включает 2 специалистов: Разработчик и Ведущий разработчик.

Модернизация программного обеспечения осуществляется силами штатных специалистов службы технической поддержки ООО «ЛАНИТ ЭКСПЕРТИЗА». Команда службы поддержки включает 2 специалистов: Разработчик и Ведущий разработчик.

5 Контакты службы технической поддержки «Banjo Framework»

Адрес: 129075, г. Москва, Мурманский проезд, д.14, корп.1, техн. эт. 2, помещ. 58

Почта: lanit_exp@lanit.ru

Телефон: +7 (495) 967-66-50 доб. 13397

Приложение 2. Описание Banjo Core (Lanit AT framework-core)

Назначение

Ядро фреймворка объединяет все те средства, которые необходимы для реализации тестов, такие как: работа с веб-драйверами, реализация паттерна Page Object, взаимодействия для тестирования Api, генерация данных, чтение конфигурации запуска. Для наиболее простого развёртывания используйте коробочную версию Banjo. В данном кратком руководстве по использованию будет рассказано о подключении данного ядра, а также вспомогательных библиотек, не включённых в ядро для большей свободы выбора тестового фреймворка, фреймворков отчётности, логгеров и т.п.

Основные особенности

Контекст

При навигации по страницам следует следить за контекстом - текущим блоком или страницей, внутри которого ищется запрашиваемый шагом элемент. Это значительно облегчает понимание, с каким именно элементом происходит работа. В связи с этой особенностью нужно явно задавать блочный элемент при помощи метода в AbstractFrameworkSteps#setCurrentBlock, внутри которого происходит дальнейшая работа (блочный элемент, наследник AbstractBlockElement или AbstractMobileBlockElement) и так же явно его освобождать при переходе к следующему блоку при помощи AbstractFrameworkSteps#resetCurrentBlock. Таким образом контекст поиска вернётся к текущей странице. Если предварительно контекст поиска не был освобождён, то поиск будет вестись внутри текущего блока и в случае успеха текущим станет найденный блок. Текущей страницей является последняя запрошенная страница.

PageCatalog

Кэш ранее запрошенных страниц. Позволяет сократить время на инициализацию страниц. В связи с этим именно PageCatalog является точкой входа для инициализации (запроса) новых страниц. Особое внимание следует уделить тому, что при запросе страницы не происходит переход на её url, а только инициализируется PageObject с прокси-объектами страницы. Конкретный поиск и получение элемента происходят в момент работы с конечным элементом.

DataGenerator

Вспомогательный утилитарный класс для генерации различных тестовых данных. В качестве примера рекомендуется реализация из коробочного фреймворка, где создание тестовых данных происходит при помощи механизма TypeRegistry Cucumber. Возможны и другие примеры реализации. Реализована генерация:

- адресов
- дат, в том числе относительных
- файлов (имитация формата только на уровне заголовка и расширения)
- чисел
- персональных данных: паспорт, свидетельство о рождении, ИНН и т.п.

- строковых данных

Именованные элементы

Для простоты обращения к элементу из Gherkin сценария его можно находить в странице и блочном элементе также по произвольному имени, заданному при помощи аннотации @WithName и метода AbstractFrameworkSteps#getElementByName. Аналогичный механизм предусмотрен для нахождения страницы при помощи библиотеки Reflections. Для этого интерфейс страницы помечается аннотацией @Title и вызывается метод AbstractFrameworkSteps#getPageByTitle.

Использование

Добавьте в зависимости к своему maven-проекту

```
<dependency>
  <groupId>com.github.lanit-izh</groupId>
  <artifactId>automation-framework-core</artifactId>
  <version>4.0.9</version>
</dependency>
```

или gradle-проекту:

```
repositories {
   mavenCentral()
   maven { url "https://jitpack.io" }
}

dependencies {
   compile("com.github.lanit-izh:automation-framework-core:4.0.9")
}
```

Для работы также потребуются дополнительные зависимости, которые могут отличаться от ваших предпочтений. Для примера мы используем Cucumber TestNG для запуска тестов и Allure для отчётности:

```
      <dependency>
         <groupId>io.cucumber</groupId>
         <artifactId>cucumber-testng</artifactId>
         <version>4.2.4</version>
      </dependency>
      <dependency>
         <groupId>io.qameta.allure</groupId>
         <artifactId>allure-cucumber4-jvm</artifactId>
         <version>2.12.1</version>
      </dependency>
```

то же для gradle

```
   compile 'io.cucumber:cucumber-testng:4.2.4'
   compile 'io.qameta.allure:allure-cucumber4-jvm:2.12.1'
```

Создайте TestNG-раннер для фреймворка в test/java:

```
@CucumberOptions(
      plugin = {"pretty",
            "io.qameta.allure.cucumber4jvm.AllureCucumber4Jvm",
```

```
            "json:target/cucumber.json",
        },
        glue = {"steps", "hooks"},
        features = "classpath:features"
)
public class TestsRunner extends AbstractTestNGCucumberTests {
}
```

В пакете test создайте подпакеты steps и hooks. В пакете test.steps будут сохраняться имплементации шагов Cucumber. В пакете test.hooks различные настроечные шаги, например шаги перед сценарием или регистрация TypeRegistry.
В пакете test.hooks создайте при необходимости файл с хуками Cucumber. Чтобы получить доступ ко всем встроенным методам, наследуйте его от AbstractFrameworkSteps. В примере ниже используются включённые в коробочную версию фреймворка утилитарные классы и расширения Atlas.

```
    @Before
    public void setUp() {
        // здесь подключаются расширения Atlas. Подробнее о них в проекте https://github.com/qameta/atlas
        Atlas atlas = getAtlas();
        atlas.extension(new ElementReadyClickSendKeysExtension());
        atlas.extension(new IsDisplayedExtension());
    }

    @After
    public void tearDown(Scenario scenario) {
        if (driverIsActive()) { // прикрепляем финальный скриншот страницы к отчёту. AllureHelper доступен в коробочной версии фреймворка

AllureHelper.attachPageSource(getDriver().getPageSource().getBytes(StandardCharsets.UTF_8));
            AllureHelper.attachScreenShot("Скриншот последней операции", getScreenShooter().takeScreenshot());
            shutdownDriver();
        }
        // Если используется Citrus для тестирования api
        MessageTracingTestListener messageTracingTestListener = (MessageTracingTestListener) getEndpointByName("messageTracingTestListener");
        messageTracingTestListener.onTestFinish(getCitrusRunner().getTestCase());
        // вывод софт-ассёртов. В коробочной версии реализовано на базе ExtendedAssert TestNG
        softAssert().assertAll();
        softAssert().flush();
    }
```

Создание PageObjects

Для создания страницы небходимо создать интерфейс в папке main/java/pages и отнаследовать интерфейс от AbstractPage для web-страниц, или от AbstractScreen для экранов мобильных приложений:

```
@Title("Стартовая страница Google")
public interface GoogleStartPage extends AbstractPage {
    @Override
    default boolean isOpen() {
        return getWrappedDriver().getCurrentUrl().matches("https?://(?:www.)?google\\.com/?.*");
    }
```

```
    @WithName("Поиск")
    @FindBy("descendant::input[@title='Поиск']")
    Input searchField();
}
```
Полностью пример доступен в коробочной версии.

Создание страниц-объектов можно посмотреть в коробочной версии фреймворка. Для упрощения реализации шагов рекомендуется придерживаться ряда правил:

- типы элементов, встречающиеся только на этой странице, должны храниться в пакете этой страницы
- рекомендуется давать конкретные и исчерпывающие названия страницы в аннотации @Title
- методы по работе с элементами страницы рекомендуется создавать внутри страницы, но если эти методы характерны для блока, который используется на нескольких страницах, логику лучше перенести в него.
- PageObject используется как абстракция "физической" страницы, однако в зависимости от контекста одной странице может соответствовать несколько PageObjects.
- для максимального переиспользования элементы необходимо именовать при помощи аннотации @WithName, либо максимально унифицировать локатор, чтобы он подпадал под все вариации типа элемента.

Создание StepDefinition

Определения шагов создаются в папке test/java/steps, но это зависит от параметров, которые были указаны в настройках Cucumber. Непосредственно создание шагов удобнее всего делать после создания feature-файла при помощи средств генерации плагина Cucumber, например для Intellij Idea. Однако, следует получившийся stepDefinition-класс наследовать от AbstractFrameworkSteps или от AbstractFrameworkAndroidSteps, это даст доступ к модулям ядра без дополнительных усилий. Рекомендуется группировать реализации шагов по конечным элементам взаимодействия. Таким образом будет проще отслеживать дубликаты и разрешать конфликты.

Создание Feature-сценариев

Если вам плохо знаком BDD подход, рекомендуется вначале ознакомиться с общими подходами. Подойдёт эта статья. Для начала создайте feature-файл в test/resources/features/. Для Intellij Idea понадобится плагин Cucumber for Java. Рекомендуется придерживаться следующих правил при создании шагов:

- шаги начинаются с маленькой буквы
- строковые значения передаются в двойных кавычках, цифровые - без
- шаги типизированы. То есть, если шаг подходит для всех элементов, логически поддерживающих данное действие, то он и должен работать со всеми элементами. Например, *кликнуть на элемент "Блок предпросмотра"* должен совершать клик по любому типу элементов, на которые только можно кликнуть. В то же время, если шаг ограничен типом *в поле "Поиск" ввести "1+1="*, такой шаг будет искать только типы, наследуемые от типа *поле* (в коробочной версии фреймворка это элементы типа Field), но никак не выпадающие списки или кнопки
- формулировка шагов должна явно указывать на тип действия:

- шаги, содержащие проверки assert, должны быть в утвердительной форме: *открыта страница "Поиск Google"* и в случае ошибки должны выбрасывать соответствующую ошибку
- шаги, выполняющие действие, должны формулироваться в виде действия: *нажать кнопку "Поиск"*
- не нужно вводить лишние слова, знаки препинания и прочие выражения, могущие создать дублирование формулировок. Например, вместо *нажать на основную кнопку "Поиск"* следует ограничиться *нажать кнопку "Основная кнопка Поиск"*, либо вообще не указывать тип там, где это не нужно для ограничения по наследованию: *нажать "Основная кнопка Поиск"*

Запуск тестов

Для запуска тестов нужно использовать maven. В зависимости от тестового фреймворка и версии Cucumber команда запуска может отличаться. В нашем случае с зависимостью от TestNG и CucumberTestNG запуск будет происходить через кукумберовский раннер:

```
mvn test -Dcucumber.options='--tags @GoogleCalc'
```

Соответствующим образом конфигурируется запуск в Jenkins или другой CI/CD.