接口自动化测试

**背景介绍**

1. 简介

功能测试、性能测试、GUI自动化回归测试已经能够满足我们的测试需求，保证网站质量，而随着产品功能越来越多、系统架构越来越复杂、新人越来越多，一些预想不到的缺陷出现在我们面前，我们必须要寻找一种更加有效的测试方法来适应当前的变化，保证产品的质量。因此接口测试应运而生。

对于Web接口应用，包含浏览器与服务器交互的HTTP协议的接口和webService接口，软件测试人员在日常的测试工作中，需要大量的手动操作来验证接口的功能。开发人员在开发过程中，需要访问其应用并且验证其功能是否正常运行，反复调试重复验证。系统维护人员也需要经常访问其应用，以确保系统的正常运行。如果某系统的接口较多，功能较为复杂，如上所述的这些操作就需要花费大量的时间和人力，如能引入自动化测试代替人工重复操作，将极大地提高团队的生产效率。在这里，我们将介绍如何使用HttpClient框架完成接口自动化测试。

1. web接口自动化测试

如今，大多数的应用软件是基于Web的应用程序并通过浏览器展示给用户并与之进行交互。不同公司和机构组织都需要测试这些应用程序的有效性。在一个高度交互性和响应的软件时代，许多组织及团队倾向于运用敏捷开发理论，自动化测试一定程度上成为了敏捷开发流程中不可或缺的手段。所谓自动化测试，就是执行自动测试工具或者用某种程序设计语言编写程序，控制被测软件中的各种模块，模拟手动测试步骤，完成测试的过程。测试自动化有很多优点，比如：频繁快速的迭代回归、高效的测试反馈、一致与重复性的执行、化繁为简的形式、弥补手工测试的可能遗漏缺陷等。目前也有许多商业和开源的软件，可辅助面向Web接口自动化测试，如：HttpClient、HttpUnit、HtmlUnit、JwebUnit等。HttpClient是一个功能丰富支持HTTP协议的客户端编程工具包，能够很好满足我们对接口的自动化测试。

**协议请求**

1. HTTP协议

HTTP协议即超文本传输协议，是一种详细规定了浏览器和万维网服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。

* **请求信息**

请求行： 例如GET /images/logo.gif HTTP/1.1，表示从/images目录下请求logo.gif这个文件。

请求头： 例如Accept-Language: en。

空行

可选消息体： 请求行和标题必须以<CR><LF>作为结尾（也就是，回车然后换行）。空行内必须只有<CR><LF>而无其他空格。在HTTP/1.1协议中，所有的请求头，除post外，都是可选的。

* **请求方法**

HTTP/1.1协议中共定义了八种方法（有时也叫“动作”）来表明Request-URL制定的资源不同操作方式。比较常用的方法有HEAD、POST、GET方法。

HEAD： 向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以在不必传

 输整个响应内容的情况下，就可以获取包含在响应消息头中的元信息。

POST： 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。

 POST请求可能会导致新的资源的建立和/或已有资源的修改。

GET： 请求获取Request-URL所标识的资源。

HEAD方法和GET方法几乎是一样的，对于HEAD请求的回应部分来说，它的HTTP头部中包含的信息与通过GET请求所得到的信息时相同的。利用这个方法，不必传输整个资源内容，就可以得到Request-URL所标识的资源信息，该方法常用于测试连接的有效性，是否可以访问，以及最近是否更新。

在本文使用的接口自动化框架中，主要使用POST和GET方法发送请求，下面对这两种方法简单做一下比较：

1. GET提交的数据会放在URL之后，以？分割URL和传输数据，参数之间以&相连，如

 EditPosts.aspx?name=test1&id=123456。POST方法是把提交的数据放到HTTP协议包的Body中。

1. GET提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限

 制。

1. GET方式提交数据，会带来安全问题，比如一个登陆页面，通过GET方式提交数据时，用户名和密

 码将出现在URL上，如果页面可以被缓存或者其他人访问这台机器，就可以从历史记录获得该用户

 的账号和密码。

* **响应信息**

在接收和解释请求消息后，服务器返回一个HTTP响应信息。HTTP响应也是由三个部分组成，分别是：状态行、响应头信息和响应正文。

状态行：包含HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器返回的响应状态码；

 Reason-Phrase表示状态代码的文本描述。

响应头信息：包括通用头、请求头、响应头和实体头四个部分。（具体内容可以去网上查阅相关内容）

响应正文：响应正文就是服务器返回的资源的内容。

1. Soap协议

接口自动化框架测试中，有些模块之间采用了Soap协议的请求方式，本节我们对Soap协议进行简单的介绍。

Soap协议即简单对象访问协议，是一种轻量的、简单的、基于XML（标准通用标记语言的一个子集）的协议，它被设计成WEB上交换结构化和固化的信息。Soap使用基于XML的数据结构和超文本传输协议（HTTP）的组合定义了一个标准的方法来使用Internet上各种不同的操作环境中的分布式对象。

* **Soap协议包含四部分**

封装：定义一个框架，该框架描述了消息中的内容是什么，谁应当处理它以及它是可选的还是必选的。

编码规则：定义了一种序列化的机制，用于交换应用程序所定义的数据类型的实例。

RPC表示：定义了用于表示远程过程调用和应答的协定。

绑定：定义了一种使用底层传输协议来完成在节点间交换封装的约定。

SOAP消息基本上是从发送端到接收端的单向传输，但它们常常结合起来执行类似于请求 / 应答的模式。所有的 SOAP消息都使用 XML 编码。一条 SOAP消息就是一个包含有一个必需的 SOAP 的封装包，一个可选的 SOAP 标头和一个必需的 SOAP 体块的 XML 文档。把 SOAP 绑定到 HTTP 提供了同时利用 SOAP 的样式和分散的灵活性的特点以及 HTTP 的丰富的特征库的优点。

* **协议结构**

Soap消息格式：

<SOAP-ENV:Envelope>

　<SOAP:HEADER>

 ……

　</SOAP:HEADER>

　<SOAP:Body>

 ……

　</SOAP:Body>

</SOAP-ENV:Envelope>

**HttpClient**

1. 简介

 HTTP 协议可能是现在 Internet 上使用得最多、最重要的协议了，越来越多的 Java 应用程序需要直接通过 HTTP 协议来访问网络资源。虽然在 JDK 的 java net包中已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，JDK 库本身提供的功能还不够丰富和灵活。HttpClient 是 Apache Jakarta Common 下的子项目，用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。HttpClient 已经应用在很多的项目中，比如 Apache Jakarta 上很著名的另外两个开源项目 Cactus 和 HTMLUnit 都使用了 HttpClient。现在HttpClient最新版本为 HttpClient 4.3 (GA) ，在接口自动化测试框架中我们使用的版本为HttpClient 4.2.5。

1. 安装

 在介绍HttpClient的功能之前，首先需要安装好HttpClient。

* HttpClient 可以在http://jakarta.apache.org/commons/httpclient/downloads.html下载。
* HttpClient 用到了 Apache Jakarta common 下的子项目 logging，你可以从这个地址http://jakarta.apache.org/site/downloads/downloads\_commons-logging.cgi下载到 common logging，从下载后的压缩包中取出 commons-logging.jar 加到 CLASSPATH 中。
* HttpClient 用到了 Apache Jakarta common 下的子项目 codec，你可以从这个地址http://jakarta.apache.org/site/downloads/downloads\_commons-codec.cgi 下载到最新的 common codec，从下载后的压缩包中取出 commons-codec-1.x.jar 加到 CLASSPATH 中。
1. 功能介绍

 HttpClient 提供的主要的功能如下，要知道更多详细的功能可以参见 HttpClient 的主页。

* 实现了所有 HTTP 的方法（GET,POST,PUT,HEAD 等）
* 支持自动转向
* 支持 HTTPS 协议
* 支持代理服务器等
1. 基本功能使用
* **GET方法**

1、创建HttpClient的实例。

 HttpClient httpClient = new HttpClient();

1. 创建GET连接方法的实例，在这里使用HttpGet方法。在HttpCet的构造函数中传入待连接的地址。

 HttpGet get = new HttpGet(url);

3、调用第一步创建好的实例的execute方法来执行第二步中创建好的method实例。

 HttpResponse httpResponse = httpClient.execute(get);

4、读取response。从返回的response中获取状态码、响应正文等信息。

5、释放连接。

 client.getConnectionManager().shutdown();

6、对得到后的内容进行处理。

* **POST方法**

POST方法用来向项目的服务器发送请求，要求它接受被附在请求后的实体，并且把它当作请求队列中请URL所指定资源的附加新子项。调用HttpClient中的HttpPost方法与HttpGet方法类似，除了设置HttpPost的实例与HttpGet有些不同之外，剩下步骤相同。

1、创建HttpClient的实例。

 HttpClient httpClient = new HttpClient();

2、创建POST连接方法的实例，在这里使用HttpPost方法。在HttpPost的构造函数中传入待连接的地址。

 HttpPost post= new HttpPost(url);

3、给post实例填充表单的值。例如一个登录接口，需要id和passwd

 NameValuePair[] data = { new NameValuePair("id", "youUserName"),new NameValuePair("passwd","yourPwd") };

4、调用第一步创建好的实例的execute方法来执行第二步中创建好的method实例。

 HttpResponse httpResponse = httpClient.execute(post);

5、读取response。从返回的response中获取状态码、响应正文等信息。

6、释放连接。

 client.getConnectionManager().shutdown();

7、对得到后的内容进行处理。

* **Soap方法**

使用soap协议的请求信息是使用xml文件传入的，使用HttpClient对这种形式请求的处理方式与post方法类似，只是在处理请求信息的时候，可以采用通过xml文件的路径以读文件流的方式将文件的内容整体作为post的表单值传入，其他步骤不变。

1. 功能深入

对接口的测试中，我们也需要考虑接口的性能是否符合预期标准，简单的同步单一请求并不能满足测试需要，所以在此我们需要引入HttpClient异步并发请求方式。我们可以使用apache HttpAsynClient 4.X编写的开源lib库，完成异步并发测试。

**框架分层**

1.简介

httpDemo框架基于HttpClient开源项目开发，使用Maven工具进行工程管理，采用TestNG

工具组织测试，应用CSV文件存储测试数据，实现测试数据与测试用例的分离，方便测试数据管理，降低自动化脚本的维护成本，实现数据驱动。

2、Maven管理

本框架所建项目是一个Maven工程项目，通过Maven管理项目所依赖的jar包，便于jar包的维护及编译打包。

3、TestNG工具

本框架使用TestNG工具组织运行test case。相对于Junit而言，TestNG的灵活的参数化配置，执行顺序组织，以及测试套件粒度的自定义划分，可更灵活的组织自动化代码。

4、Data Driven

参数化是自动化测试框架设计的关键。本框架采用modules和demo的分层设计，使用csv文件存储测试数据，实现数据驱动。

5、日志收集

自动化执行过程的日志信息，对于失败用例的分析定位以及全过程的跟踪记录是十分重要的。相对于简单的输出打印，本框架集成了主流的日志收集工具log4j，通过配置log4j.properties文件，定义日志级别内容及日志输出路径收集日志信息。

使用指南

1. 工程结构

* **base Package**

基类，定义接口测试中必须要执行的操作：执行认证授权请求、清理环境和释放httpClient连接等。

* **common Package**

公共类，可根据不同产品的业务特点设计公共通用的业务处理方法。

* **request Package**

HttpClientDriver初始化HttpClient对象。

RequestDriver类定义了对请求方式、响应信息、设定cookie等的处理方法。

* **support Package**

对接口自动化起辅助作用的方法的实现，如使用异步并发方式完成压力测试的AsyClient类、操作数据库的JdbcExecutor类等。

* **modules Package**

 存放由.csv文件自动生成的同名实体类。

* **testDemo Package**

存放由.csv文件生成的具体的测试demo。

* **tools Package**

框架中用到的一些工具类。

* **testData Package**

 做压力测试或使用其他功能时可能会用到的数据，为了便于管理，单独存放到一个package中。

* **lib**

存放项目工程外部依赖的jar文件、dll文件等。

* **configs**

存放配置文件，如：CONFIG.properties、log4j.properties、testng.xml。

* **logs**

存放生成的日志文件。

* **csvDatas**

存放页面数据文件csv文件。

* **sqlDatas**

存放测试用到了清理数据库的.sql文件。

* **xmlDatas**

存放请求数据xml文件。

2. 数据文件解析

* + **CSV文件**

 CSV文件的文件名表示具体的模块名，系统的每个模块对应一个CSV文件。

CSV文件的默认为：featureName|name|typeAndUrl|requireData|expectResponse：模块名称、接口名称、请求类型及action、请求信息（如果是soap请求，该字段为null，数据从xml文件中读取）、期望返回数据，并用“|”做分割。requireData中要写成一个标示符对应一个数据的形式，中间要以”<n>”作为分割，若存在多条数据，每条数据建使用”<d>”作为分隔。例如：要对登录接口进行测试，那CSV文件中的数据组织成以下形式

* + **XML文件**

 Soap使用基于XML的数据结构和超文本传输协议（HTTP）的组合定义了一个标准的方法来使

 用Internet上各种不同的操作环境中的分布式对象。在工程中将soap请求信息呀xml文件的形式

 存放，在执行用例时，如果碰到请求是Soap关键字时，系统就自动会到xmlDatas文件夹下找到指

 定文件，xmlDatas文件夹下的命名方式为：xmlDatas/featureName/name.xml。

* + **JAVA文件**

对于CSV文件，需要将它生成相应的Java文件，通过这个中间桥梁，才能识别所存储的数据。将生成的Java文件导入至modules Package中。

* + **SQL文件**

 在测试过程中，我们往往需要准备一些数据，便于后期使用。例如，如果查询一个用户就

 必须保证该用户已经在数据库中存在，所以我们提前需要向数据库中手动插入这样一条信息，

 sqlDatas文件夹下的sql文件就起到了这样的作用，sql文件中存放的是直接操作数据库的sql语句。

 sqlDatas文件夹下的命名方式为：sqlDatas/featureName/name/xxx.sql，在测试之前首先遍历该文件

 夹，看某个模块下的action是否含有.sql文件，如果有，就执行初始化或者清理数据库的操作，否

 则就跳过此步。

3. 配置文件

* **CONFIG**

存储在…\resources\config中的CONFIG.properties文件是框架的配置文件。

**URL**：要请求的服务器主机地址及端口号。

**Domain**：表示cookie所在的域，默认为请求的地址。

**Path：**表示cookie所在的目录，默认为根目录/。

**JSESSION\_NAME**：cookie的JSESSIONID,每次登陆只要不重新登陆该值不会改变。

**ExpectStatusCode：**期望返回响应的状态码。

* **Maven Pom**

通过配置POM.XML文件，以达到项目管理目的。

<artifactId>testng</artifactId>

<name>httpDemo</name>

 该标签定义的是项目导入到Eclipse后的名称。

 <suiteXmlFile>resources/config/TestNG.xml</suiteXmlFile>

由于框架用TestNG管理，所以需要配置TestNG的XML文件，该标签定义的是TestNG的XML文件的路径。

* **Testng.xml**

由于框架用TestNG组织TestCase运行，所以需要配置TestNG的XML文件，打开对应的XML文件，配置如下内容：

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Suite" parallel="false">

 <test name="Test">

 <classes>

 <class name="com.sugon.qa.iTest.testDemo"/>

 </classes>

 </test>

</suite>

 其中的class标签中的name属性为需要被执行的Java文件，如上述举例即为，需要运行在iTest文件夹下的testDemo类下的所有含有Demo的方法，如果需要配置多条要被执行的Java文件，也请同样在class后面声明。

4. 用例的执行逻辑

 **第一步：生成modules和demo文件。**

 正常情况下，测试用例（demo文件）是需要测试人员手动输入的，然而当接口较多，这个工作就显得比较复杂且容易出错，为了避免上述问题，在框架中我们使用工具类实现由CSV文件自动生成测试用例和modules文件的功能，在运行测试用例之前，测试人员需要运行csvTojava Package中的Test\_PageClassGenerator.java java工程，然后刷新modules Package 和testDemo Package文件夹，就会看到modules文件和demo用例分别生成到两个文件夹中（每一个csv文件对应一个modules实体类和一个测试demo）。

**第二步：运行测试用例。**

testDemo Package文件夹中的测试用例生成无误后，就可以运行测试用例，运行方法可以是单独运行某个测试用例或者直接运行httpDemo这个工程，运行所有测试用例。

 下面是由csv文件生成的一条测试用例代码：

**package** com.sugon.qa.iTest.testDemo;

**import** org.apache.log4j.Logger;

**import** org.testng.annotations.BeforeMethod;

**import** org.testng.annotations.Test;

**import** com.sugon.qa.iTest.base.TestBase;

**public** **class** RequestTestDemo **extends** TestBase{

 //实例化logger对象

 **protected** **static** Logger *logger* = Logger.*getLogger*(RequestTestDemo.**class**);

 @BeforeMethod

 **public** **void** beforeMethod() {

 commonFeature = (CommonFeature)helper.getObj("CommonFeature");

 }

 @Test

 **public** **void** httpTestDemo1()**throws** Exception {

 *logger*.info("执行创建测试数据数据sql");

 commonFeature.*account*.initSql();

 *logger*.info("执行account action请求");

 commonFeature.*account*.httpResquest();

 *logger*.info("断言验证account action请求响应状态码");

 commonFeature.*account*.AssertResponseCode();

 *logger*.info("断言验证account action请求响应内容");

 commonFeature.*account*.AssertResponseEntity();

 *logger*.info("执行清理数据sql");

 commonFeature.*account*.cleanSql();

 }

 @Test

 **public** **void** httpTestDemo2()**throws** Exception {

 *logger*.info("执行account action请求");

 }

}

 上述用例是由csv文件自动生成的，可以测试模块中的一个或多个接口，要执行这条用例时可以直接在eclipse IDE中点击鼠标右键，选择run as→TestNG test 就可以看到用例执行；如果想要运行整个工程中的用例，在工程名附近点击鼠标右键，同样选择run as→TestNG test执行所有测试demo。