目录 (五十七期·上)

每次不重样，教你收获最新测试技术！

微信扫一扫关注我们

投稿邮箱：editor@51testing.com

Java+TestNG+Jenkins 实现接口(API)自动化测试全教程

◆ 作者：王东

第一章、概述

随着移动互联网的逐渐普及，越来越多的人在手机中安装和使用 APP，APP 的用户体验对于其厂商来说就显得尤为重要。有个常识大家肯定知道：APP 一般是通过 HTTP/HTTPS 协议的接口来获取所需的数据和相应的业务逻辑，所以接口测试的重要性越来越凸显。

由于当前市场竞争的加剧，很多 APP 发版周期变得越来越短，3-4 天就发一次版也很平常了。这种情况下，自动化测试就体现出其价值和优势来了。相比手工测试，自动化测试快速高效，测试结果可靠可信，不会因不同人员测试而给出不同的测试结果。

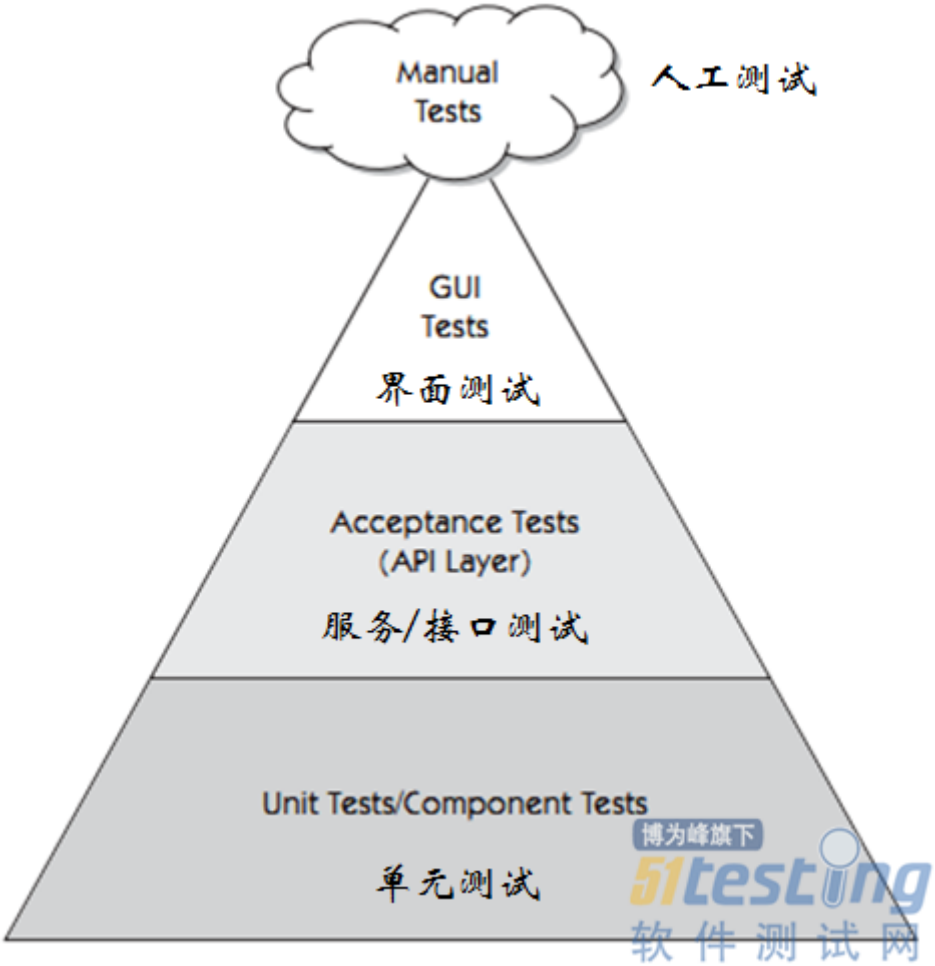
说起自动化测试，可能很多人首先想到的就是模拟人工测试的 GUI 自动化测试。但总结本人近些年的测试实践，个人认为接口自动化测试比 GUI 自动化测试有更大的意义和重要性。

相比 GUI 自动化测试，接口自动化测试主要有以下三个优势:

1.构建成本低：只需要根据给出的参数，执行后判断是否与预期结果一致即可；而 GUI 自动化测试则需要安装和配置较为复杂的测试环境，其学习成本和学习曲线也更高;

2.稳定性高：接口测试针对的是 API，没有图形界面，不会受到图形界面中元素变化因素的困扰，所以测试脚本相对比较稳定；而 GUI 自动化测试针对的是图形界面，而这部分变化是较频繁的，这也意味着对应的 GUI 自动化测试脚本经常需要修改以适应新的图形界面；另外，GUI 自动化测试技术本身就比接口自动化技术更为复杂，GUI 自动化测试过程中可能会经常出现一些莫名其妙的错误，其实这也很容易理解，接口测试毕

竟属于相对底层的测试，复杂性较低，当然出错的概率也低了，来看下著名的测试金字塔大家就更容易理解了；



3.速度快：执行一个接口自动化用例脚本可能不用一秒钟，但执行一个 GUI 自动化用例脚本至少需要 5 秒以上，当用例数量增加时，这种差别将非常明显。

所以多开展接口测试，并且结合现在广为使用的持续集成工具 Jenkins 实现接口测试的全过程自动化无疑是一个值得探索和尝试的自动化测试实践。

第二章、HttpClient 和 TestNG 简介

因为本人更熟悉和偏爱 Java，所以选用了 Java 语言作为接口用例脚本编写的语言。虽然现在很多人使用 Python 语言编写自动化测试脚本，但个人觉得 Python 语言不如 Java 语言严谨(Python 是弱类型，Java 是强类型)，出现变量类型错误不容易被发现，当代码量较大时(大于 1000 行以上)，使用 Java 无疑效率更高(呵呵，欢迎反对的同学来喷)。

因为选择了 Java，所以采用了 Apache 的开源框架 HttpClient 作为接口调用的底层框架。

HTTP 协议是现在 Internet 上使用得最多、最重要的协议了，越来越多的 软件需要通过 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 和浏览器有点像，但却不是浏览器。它是一个 HTTP 通信库，因此它只提供一个通用浏览器应用程序所期望的功能子集，最根本的区别是 HttpClient 中没有用户界面。HttpClient 只能以编程的方式传输和接受 HTTP 消息。

HttpClient 的主要功能:

1.实现了所有 HTTP 的方法（GET、POST、PUT、HEAD、DELETE、HEAD、OPTIONS、CONNECT 和 PATCH）;

2.支持 HTTPS 协议;

3.支持代理服务器（Nginx 等）;

4.支持自动（跳转）转向.

在测试框架的选择上，果断选择了 TestNG，虽然本人 JUnit 也用过，但确实 TestNG 的功能比 JUnit 强大许多。为啥？因为 TestNG 就是在 JUnit 的基础上发展起来的啊。

第三章、在 Eclipse 中新建一个 Maven 项目

3.1 必备工具安装

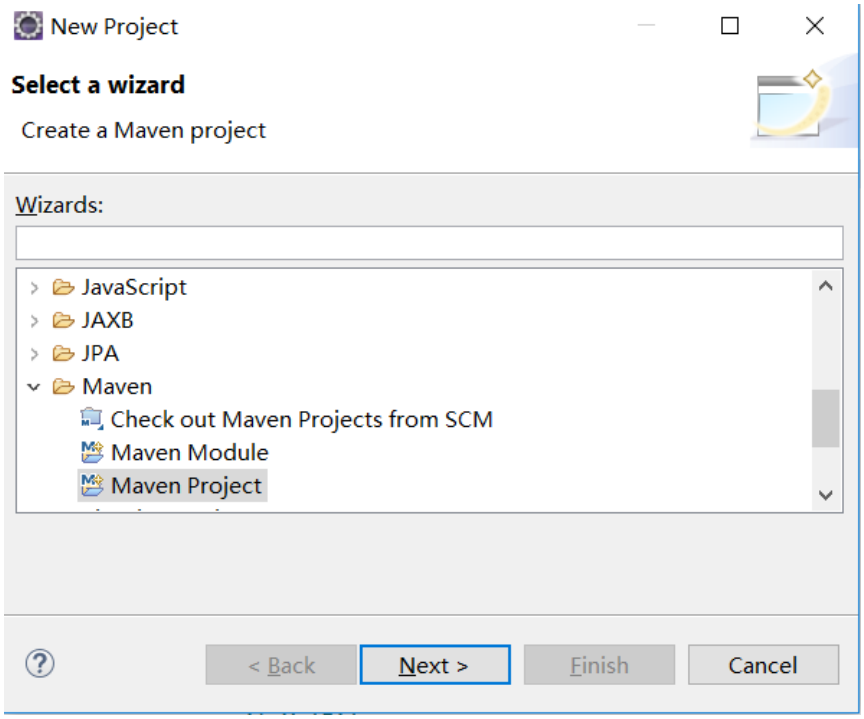
3.1.1 JDK: 使用 Java 编程，JDK 必须安装和配置好。建议下载和安装 JDK 1.8, 网上有很多相关资料和教程，这里略过。

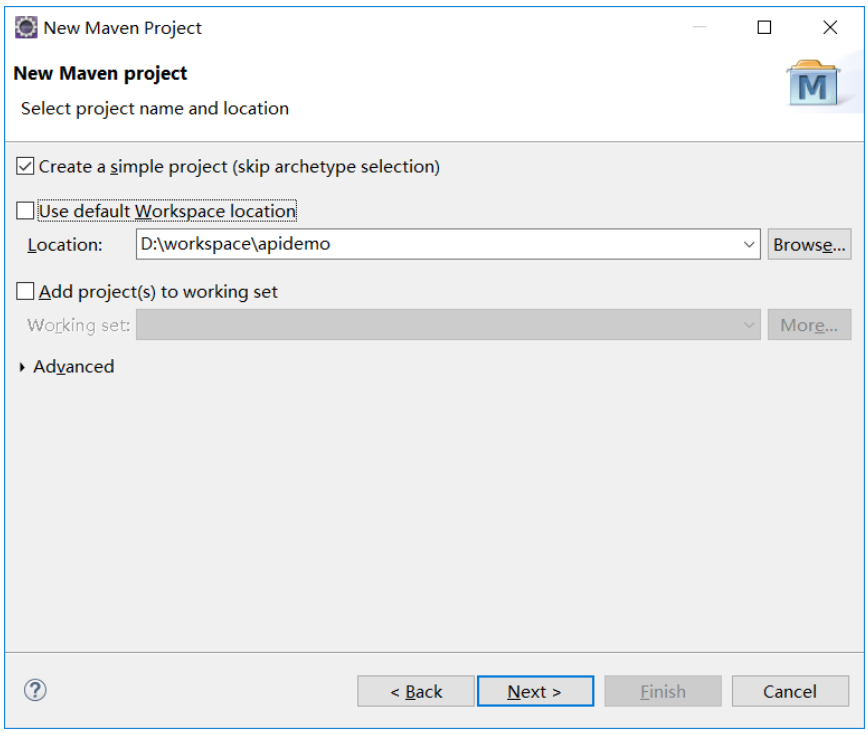
3.1.2 Eclipse: 进行 Java 编程的 IDE(集成开发环境),能大幅度提高工作效率。建议安装最新版本(最新版本应该已经包含了 Maven 插件，无需在 Eclipse 中再安装和配置 Maven 插件)。网上有很多相关资料和教程，这里略过。

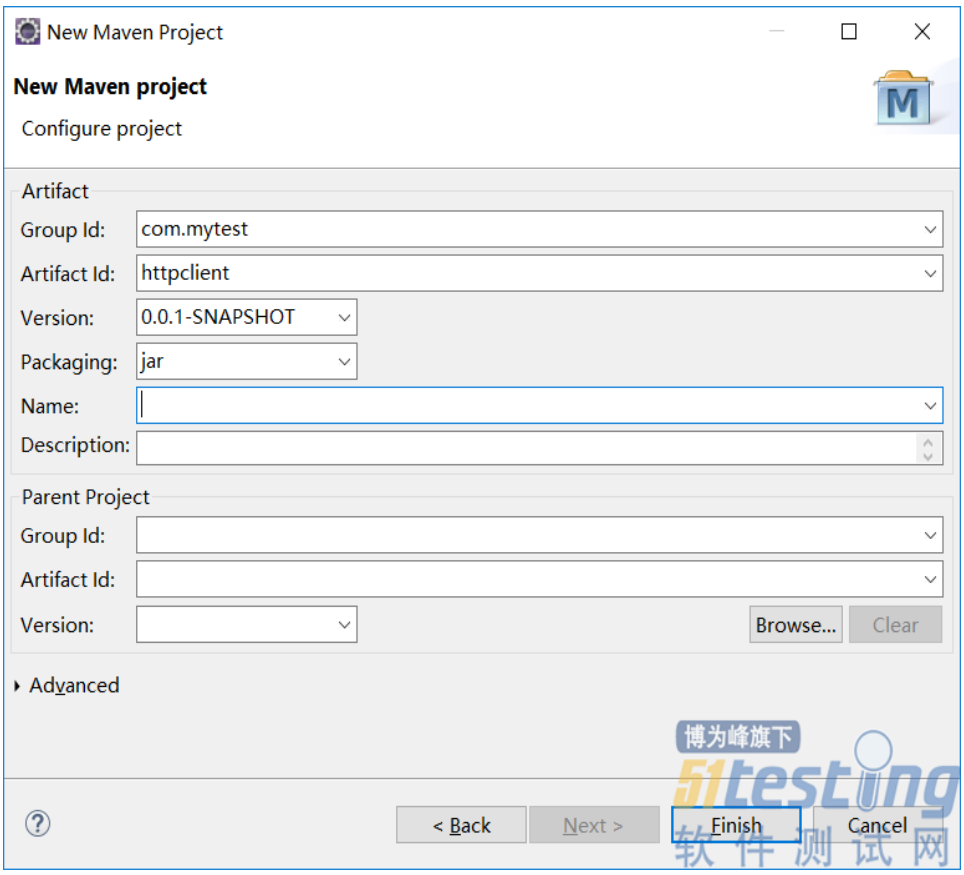
3.1.3 Maven：可自动下载 Java 项目相关的 Jar 文件，自动打包测试等，是 Java 编程人员必备神器。建议安装最新版本，网上有很多相关资料和教程，这里略过。

3.2 在 Eclipse 中新建一个 Maven 项目

双击 Eclipse 图标，然后新建一个 Maven 项目。







在自动生成的 pom.xml 文件中加入以下片段:

```
<dependencies>
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.1.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.httpcomponents</groupId>
        <artifactId>httpclient</artifactId>
        <version>4.5.11</version>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>fastjson</artifactId>
        <version>1.2.62</version>
    </dependency>
</dependencies>
```

添加完毕后保存，就可以看到 eclipse 会在后台开始自动下载并添加以上项目依赖的 jar 包。

我们现在主要引入的是 httpclient 以及 testNG 两个 Jar 包：

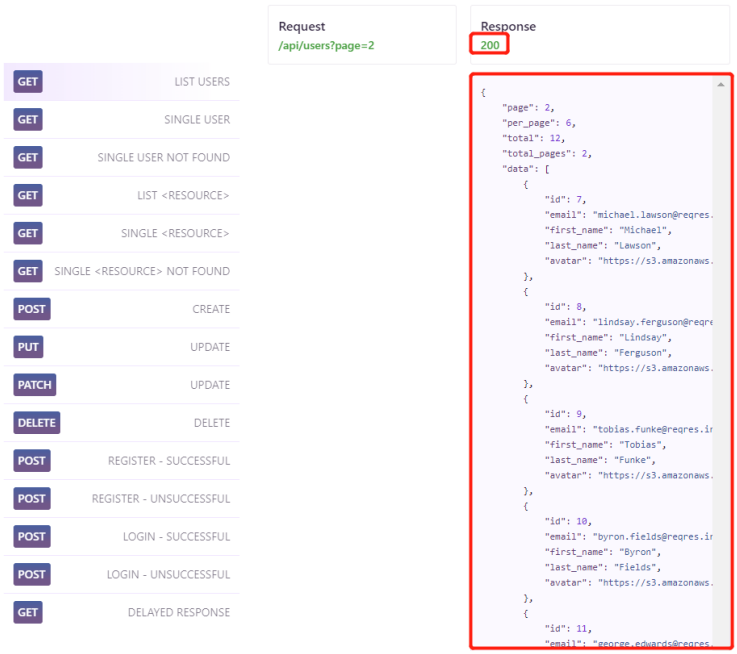
- httpclient 是一个高效的、功能丰富的支持 HTTP 协议的客户端编程工具包，我们用它来创建和管理请求；
- testNG 是一套单元测试框架，为我们提供测试入口和测试组织；
- fastjson 是阿里巴巴推出的 JSON 对象创建和解析包，据说是当前速度最快的一个。

后续如果用到其他依赖包我们再去添加相应的 xml 片段到 pom.xml 文件 dependencies 标记中，可以通过 https://mvnrepository.com/网站来找到所需引入的 jar 文件的对应的<dependency>片段。

第四章、Postman 简介

4.1 确定待测接口

我们需要有待测的接口，使我们可以发出请求并能正确返回请求结果。我们选用 https://reqres.in/这个网站作为我们接口测试的网站。这个网站提供了常用的 Get,Post 接口，也有 Put，Delete 等接口，从下图可以看到，我们可以对这个接口进行符合标准格式的请求，红框中给出的就是我们要去验证的响应信息，所以很适合我们练习选用。



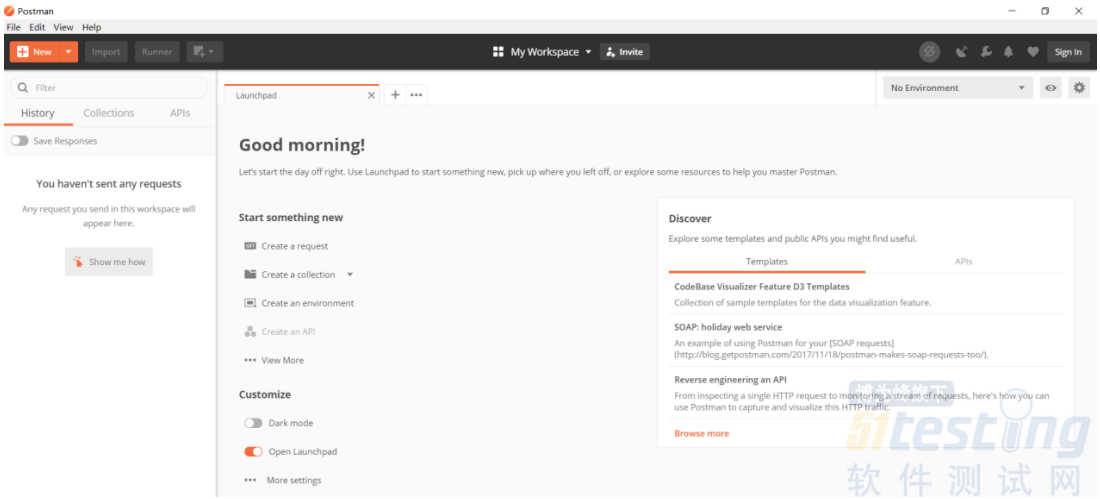
4.2 Postman 简介

Postman 是一款很好用的 API/接口功能测试工具，支持所有类型的 HTTP 请求方法，操作简单方便。有免费版、pro 版、企业版三个版本。个人学习及日常工作免费版完全够用。

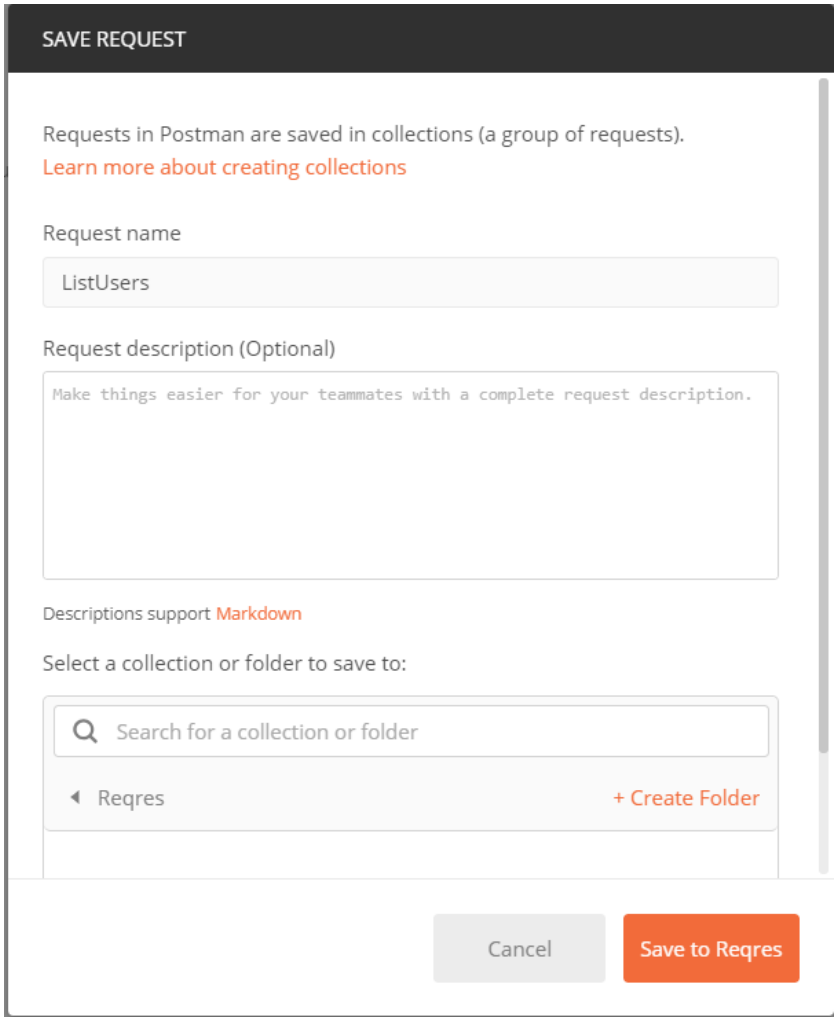
下载地址：https://www.getpostman.com/

4.3 使用 Postman 发送请求及查看响应

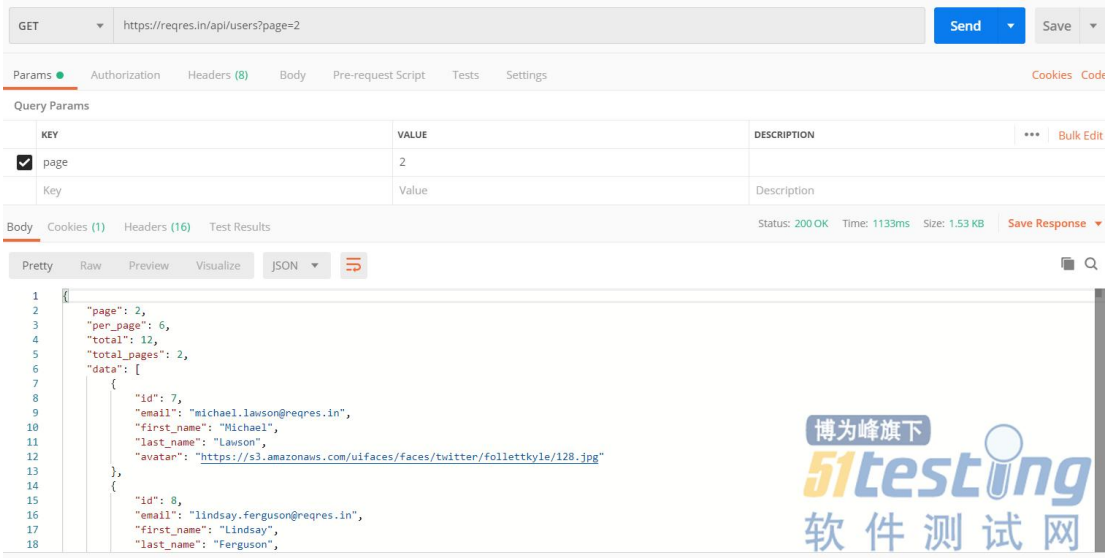
下载后安装好 Postman，然后双击图标打开，我们看到了 Postman 的主界面，应该是类似这样:



然后我们新建一个 Get 请求，点击左上角的 New 图标，然后选中 Create New Request，在弹出的窗口中输入 Request Name，必须新建一个 Collection 并选择该 Collection(集合，这里也可理解为项目)用于存放一组相关的请求或来自同一个站点的接口，最后点击保存成功创建该请求。



之后在 Get 右侧输入接口 URL，点击 Send 按钮即可发送请求并看到响应结果了。



和 https://reqres.in/网站的对应接口响应结果对照下，可以确认 Postman 执行请求后返回的响应结果完全正确。

第五章、Get 方法的接口测试

5.1、创建发送接口的测试类

我们的项目需要有这样一个类，它能够实现发送请求，接收响应，验证响应的功能。暂时我们只考虑发送 Get 方法的请求。

5.1.1 创建所需变量

在我们的第一个测试类中，我们需要使用 httpClient 来发送请求，接收响应，然后对响应信息做一个验证。

在我们的项目 src/test/java 目录下新建一个包名为：com.myteest.httpclient.test，在包下新建一个普通类:SimpleGetTest

首先我们考虑需要如下变量:

```
String url;
CloseableHttpClient httpClient;
 CloseableHttpResponse response;
 HttpEntity responseBody;
    int responseCode;
```

- url 是我们进行 get 请求的地址;
- client 是用来发送 http 请求的 CloseableHttpClient 对象的实例;
- response 用来存储我们接收到的响应的 CloseableHttpResponse 对象的实例;
- responseBody 用来存储响应的主体信息;
- responseCode 用来存储响应的状态码;

5.1.2 利用 httpclient 框架编写代码实现请求发送和响应接收

接下来实现请求的发送和响应接收。

我们就以网站上的第一个 get 请求为例，将其网址和 Request 部分复制并连接起来，形成 URL 如下:

```
url = "https://reqres.in/api/users?page=2";
```

接下来用三行代码来发送请求，接收响应:

```
//创建一个 httpClient 的实例
      client = HttpClients.createDefault();
```

```
        //创建一个 httpGet 请求实例
        HttpGet httpGet = new HttpGet(url);
        //使用 httpClient 实例的 execute 方法发送刚创建的 get 请求，并用 httpResponse 将响应接收
        response = client.execute(httpGet);
```

其实到这一步我们的主体工作已经做完了，接下来要对接收到的响应进行一个处理、分析和验证。

首先来看下状态码如何获取:

```
responseCode = response.getStatusLine().getStatusCode();
```

其次从响应中取得响应主体并转为字符串，最后使用 JSON 对象解读该字符串，返回一个 JSONOBject 对象，利用该对象获取指定的键值:

```
// 从响应中提取出响应主体
        responseBody = response.getEntity();
        //转为字符串
        String responseBodyString = EntityUtils.toString(responseBody, "utf-8");
        // 创建 Json 对象，把上面字符串序列化成 Json 对象
        JSONObject responseJson = JSON.parseObject(responseBodyString);
        // json 内容解析
        int page = responseJson.getInteger("page");
```

完整代码如下:

```
package com.mytest.httpclient.test;
import java.io.IOException;
import org.apache.http.HttpEntity;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
public class SimpleGetTest {
  String url;
  CloseableHttpClient client;
  CloseableHttpResponse response;
  HttpEntity responseBody;
  int responseCode;
  public void sendGet() throws Throwable {
        String url = "https://reqres.in/api/users?pages=2";
        // 创建一个 httpClient 的实例
        client = HttpClients.createDefault();
```

```
            // 创建一个 httpGet 请求实例
            HttpGet httpGet = new HttpGet(url);
            try {
                // 使用 httpClient 实例发送刚创建的 get 请求，并用 httpResponse 将响应接收
                response = client.execute(httpGet);
                // 从响应中提取出状态码
                responseCode = response.getStatusLine().getStatusCode();
                System.out.println(responseCode);
                // 从响应中提取出响应主体
                responseBody = response.getEntity();
                // 转为字符串
                String responseBodyString = EntityUtils.toString(responseBody, "utf-8");
                System.out.println(responseBodyString);
                EntityUtils.consume(responseBody);
            } catch (ClientProtocolException cpe) {
                cpe.printStackTrace();
            } catch (IOException ioe) {
                ioe.printStackTrace();
            }finally {
                client.close();
                response.close();
            }
        }
        public static void main(String[] args) throws Throwable {
            SimpleGetTest simpleGetTest = new SimpleGetTest();
            simpleGetTest.sendGet();
        }
    }
```

第六章、初步封装

在上一章中，我们写了第一个 get 请求的测试类，这一章我们来对它进行初步优化和封装。

6.1、分离请求发送类

首先想到的问题是，以后我们的接口自动化测试框架会大量用到发送 http 请求的功能。

那么这一部分的处理，可以将它分离出来，以后的测试类只需要调用请求类的方法实现发送请求和接收响应。

在我们的项目目录 src/main/java 下，新建一个包名为 com.mytest.httpclient，在包下新建一个类，名称为 HttpClienUtil。

这个类我们把上一章写的发送请求和处理反馈的代码迁移过来并进行了重构:

```
package com.mytest.httpclient;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;
import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
public class HttpClienUtil {
  private CloseableHttpClient httpClient;
  private CloseableHttpResponse response;
  private RequestConfig requestConfig;
  public String HTTPSTATUS = "HttpStatus";
  public HttpClientUtil() {
requestConfig =
RequestConfig.custom().setConnectTimeout(5000).setConnectionRequestTimeout(1000).setSocketTime
out(10000).build();
  }
  /**
   *
   * @param connectTimeout                设置连接超时时间，单位毫秒。
```

```
    * @param connectionRequestTimeout 设置从 connect Manager(连接池)获取 Connection
    *                                 超时时间，单位毫秒。这个属性是新加的属性，因为
目前版本是可以共享连接池的。
    * @param socketTimeout            请求获取数据的超时时间(即响应时间)，单位毫秒。
    *                                 如果访问一个接口，多少时间内无法返回数据，就直
接放弃此次调用。
    */
  public HttpClientUtil(int connectTimeout, int connectionRequestTimeout, int socketTimeout) {
requestConfig = RequestConfig.custom().setConnectTimeout(connectTimeout)
.setConnectionRequestTimeout(connectionRequestTimeout)
.setSocketTimeout(socketTimeout).build();
  }
  public JSONObject sendGet(String url, HashMap<String, String> params, HashMap<String, String>
headers)
            throws Exception {
        httpClient = HttpClients.createDefault();
        // 拼接 url
        if (params != null) {
            List<NameValuePair> pairs = new ArrayList<NameValuePair>();
            for (Map.Entry<String, String> entry : params.entrySet()) {
                String value = entry.getValue();
                if (!value.isEmpty()) {
                    pairs.add(new BasicNameValuePair(entry.getKey(), value));
                }
            }
            url += "?" + EntityUtils.toString(new UrlEncodedFormEntity(pairs), "UTF-8");
        }
        HttpGet httpGet = new HttpGet(url);
        try {
            httpGet.setConfig(requestConfig);
            // 加载请求头到 httpget 对象
            if (headers != null) {
                for (Map.Entry<String, String> entry : headers.entrySet()) {
                    httpGet.setHeader(entry.getKey(), entry.getValue());
                }
            }
            response = httpClient.execute(httpGet);
            // 获取返回参数
            HttpEntity entity = response.getEntity();
            String result = null;
            if (entity != null) {
                result = EntityUtils.toString(entity, "UTF-8");
```

```
                }
                // 释放请求，关闭连接
                EntityUtils.consume(entity);
                JSONObject jsonobj = JSON.parseObject(result);
                jsonobj.put(HTTPSTATUS, response.getStatusLine().getStatusCode());
                return jsonobj;
            } finally {
                httpClient.close();
                response.close();
            }
        }
        public JSONObject sendGet(String url, HashMap<String, String> params) throws Exception {
            return this.sendGet(url, params, null);
        }
        public JSONObject sendGet(String url) throws Exception {
            return this.sendGet(url, null, null);
        }
    }
```

代码重构后，增加了两个构造函数，一个使用默认值，一个可以给其传递相应的参数，都是用于实例化 RequestConfig 对象并设置请求的超时时间；也重载了两个 sendGet 方法，都是用于发送 get 请求并获取 JSONObject 对象，为了使用方便可以使用只有一个或两个参数的 sendGet 方法。

请注意看这行代码:

```
jsonobj.put(HTTPSTATUS, response.getStatusLine().getStatusCode());
```

这里将响应对应的状态码(status code)也压入了 JSONObject 对象，是为了方便直接通过 JSONObject 对象取得响应状态码。

后续我们在测试类里面，直接调用该类的这些方法。

6.2、引入 JSON 解析工具类

为了对响应结果进行解读和验证，我们使用 com.alibaba.fastjson 包中的 JSON 对象的 parseObject 方法和 JSONObject 对象的相应的获取键值的方法 get+数据类型，请参见以下代码:

```
responseBody = response.getEntity();
//转为字符串
String responseBodyString = EntityUtils.toString(responseBody, "utf-8");
// 创建 Json 对象，把上面字符串序列化成 Json 对象
```

```
JSONObject responseJson = JSON.parseObject(responseBodyString);
// json 内容解析
int page = responseJson.getInteger("page");
```

但由于返回的 JSON 字符串并不总是这么简单的格式，有时会有 JSON 数组以及嵌套的 JSON 字符串，所以我们单独写一个类用来解读 JSON 对象。

下面是这个 get 请求对应的响应 JSON 数据截图

上图第一个红框”page”是一个 JSON 对象，我们可以根据键”page”来找到对应的值是 2，而第二个红框“data”是一个 JSON 数组，并不是一个 JSON 对象，不能直接去拿到里面的值，需要遍历数组；第三个红框”first_name”是在一个嵌套 JSON 字符串中，可以根据里层 JSON 对象的” first_name”来找到对应的值”Michael”。

下面，我们写一个 JSON 解析的工具方法类，如果是像第一个红圈的 JSON 对象，我们直接返回对应的值，如果是需要解析类似 data 数组里面的 JSON 对象的值，我们构造方法默认解析数组第一个元素的内容。

在项目目录 src/main/java 的包 com.mytest.httpclient 下，新建一个类文件，名称为 Util，代码如下：

```
package com.mytest.httpclient;
import com.alibaba.fastjson.JSONArray;
```

```
import com.alibaba.fastjson.JSONObject;
public class Util {
  /**
   *
   * @param responseJson ,这个变量是拿到响应字符串通过 json 转换成 json 对象
   * @param jpath,这个 jpath 指的是用户想要查询 json 对象的值的路径写法  jpath 写法举例:
   * 1) per_page
   * 2)data[1]/first_name，data 是一个 json 数组，[1]表示索引
   * /first_name 表示 data 数组下某一个元素下的 json 对象的名称为 first_name
   * @return，返回 first_name 这个 json 对象名称对应的值
   */
  public static String getValueByJPath(JSONObject responseJson, String jpath) {
        Object obj = responseJson;
        for (String s : jpath.split("/")) {
            if (!s.isEmpty()) {
                if (!(s.contains("[") || s.contains("]"))) {
                    obj = ((JSONObject) obj).get(s);
                } else if (s.contains("[") || s.contains("]")) {
                    obj = ((JSONArray) ((JSONObject) obj).get(s.split("\\[")[0]))
                            .get(Integer.parseInt(s.split("\\[")[1].replaceAll("]", "")));
                }else if (s.contains("{") || s.contains("}")) {
                    obj = ((JSONArray) ((JSONObject) obj).get(s.split("\\{")[0]))
                            .get(Integer.parseInt(s.split("\\{")[1].replaceAll("}", "")));
                }
            }
        }
        return obj.toString();
  }
}
```

简单解释下上面的代码，主要是查询三种 json 对象的的值:

第一种最简单的，这个 json 对象在整个 json 串的第一层，例如上面截图中的 per_page，这个 per_page 就是通过 jpath 这个参数传入，返回的结果就是 2；第二种 jpath 的查询，例如我想查询 data 下第一个用户信息里面的 first_name 的值，这个时候 jpath 的写法就是 data[0]/first_name，查询结果应该是 Michael;

第三种是嵌套 JSON 查询，例如下方的 JSON 字符串，里面有两个嵌套的大括号，如果想取得 first_name 的值，这个时候 jpath 的写法就是 data/first_name。

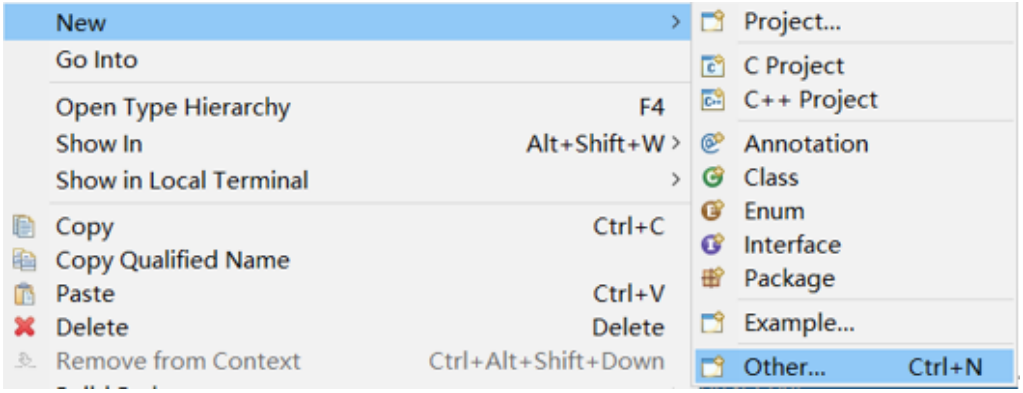
第七章、TestNG 断言

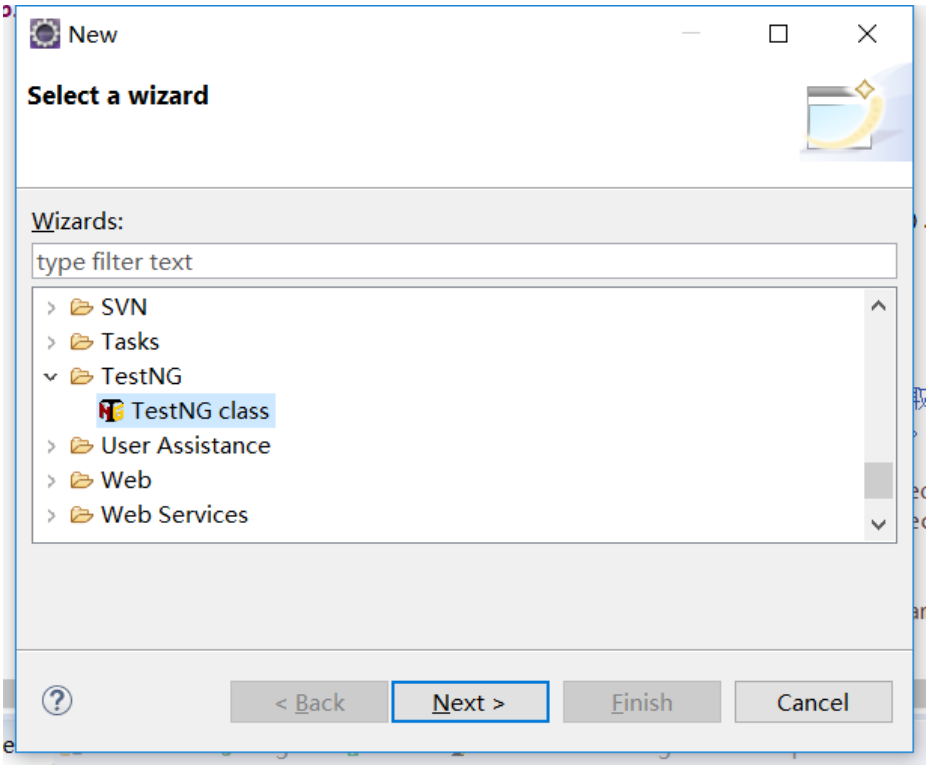
7.1、引入 TestNG

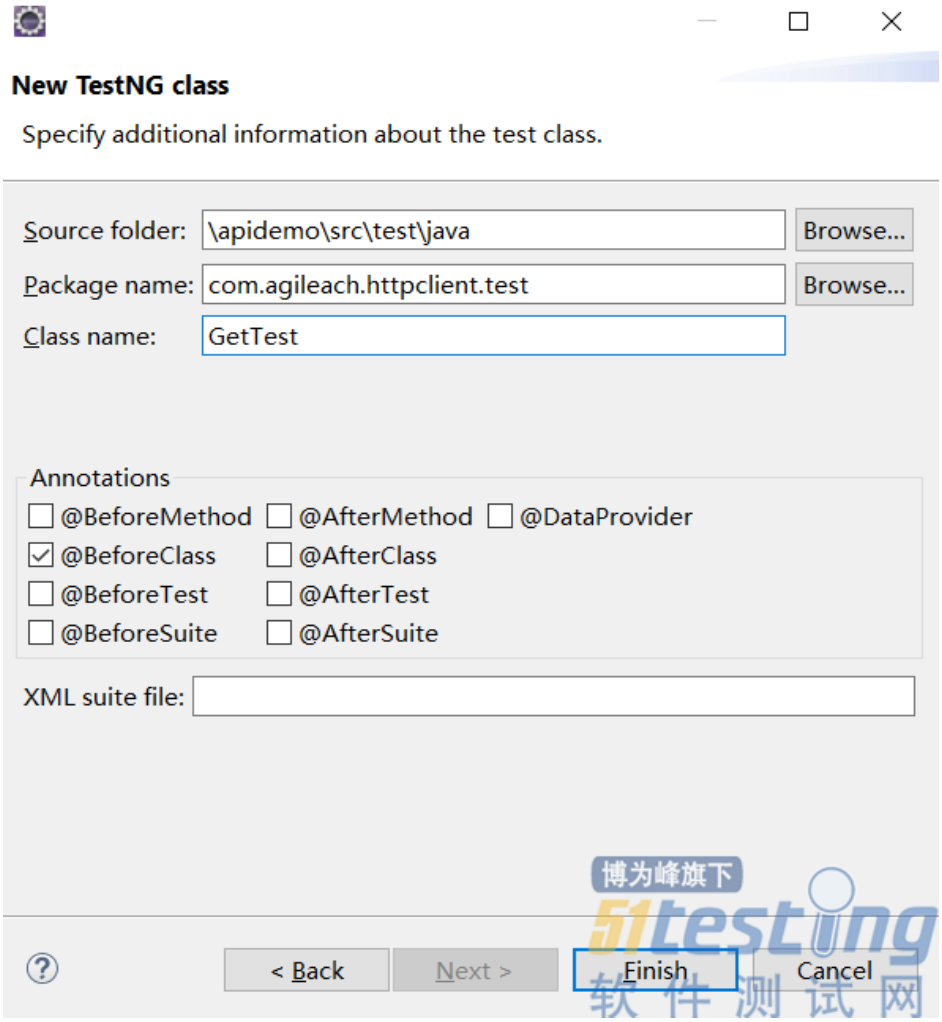
上一章中我们还没有很好的手段来执行测试和验证结果，这里我们引入 TestNG 来帮助完成这部分功能。

7.1.1 创建 TestNG 测试类

在项目目录 src/test/java 下的包 com.mytest.httpclient.test 下新建一个支持 testNG 的类，类名为 testGet。




New
Select a wizard
Wizards:
type filter text
SVN
Tasks
TestNG
TestNG class
User Assistance
Web
Web Services
< Back
Next >
Finish
Cancel


New TestNG class
Specify additional information about the test class.
Source folder: \apidemo\src\test\java
Browse...
Package name: com.agileach.httpclient.test
Browse...
Class name: GetTest
Annotations
@BeforeMethod
@AfterMethod
@DataProvider
@BeforeClass
@AfterClass
@BeforeTest
@AfterTest
@BeforeSuite
@AfterSuite
XML suite file:
< Back
Next >
Finish
Cancel

写入以下测试代码:

```
package com.mytest.httpclient.test;
import java.io.IOException;
import org.apache.http.HttpEntity;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.testng.Assert;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import com.mytest.httpclient.Util;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
public class GetTest {
  CloseableHttpClient client;
  CloseableHttpResponse response;
  HttpEntity responseBody;
  int responseCode;
  @BeforeTest
  public void setUp() {
        // 创建一个 httpClient 的实例
        client = HttpClients.createDefault();
  }

  @Test
  public void getTest() throws ClientProtocolException, IOException {
        String   url = "https://reqres.in/api/users?pages=2";
        // 创建一个 httpGet 请求实例
        HttpGet httpGet = new HttpGet(url);
        // 使用 httpClient 实例发送刚创建的 get 请求，并用 httpResponse 将响应接收
        response = client.execute(httpGet);
        // 从响应中提取出状态码
        responseCode = response.getStatusLine().getStatusCode();
        Assert.assertEquals(responseCode, 200, "The response code should be 200!");
        // 从响应中提取出响应主体
        responseBody = response.getEntity();
        //转为字符串
        String responseBodyString = EntityUtils.toString(responseBody, "utf-8");
        // 创建 Json 对象，把上面字符串序列化成 Json 对象
```

```
        JSONObject responseJson = JSON.parseObject(responseBodyString);
        // json 内容解析
        int page = responseJson.getInteger("page");
        Assert.assertEquals(page, 2, "The page value should be 2!");
        String firstName = Util.getValueByJPath(responseJson, "data[0]/first_name");
        Assert.assertEquals(firstName, "Michael", "The first name should be Michael!");
    }
}
```

这里我们在 beforeTest 里去创建一个 HttpClient 的实例，beforeTest 方法是 TestNG 内置的方法，它会在测试方法执行之前被执行，所以可以在该方法中写入一些初始化的代码，然后在 getTest 方法中调用上一章中封装好的方法获取响应主体和状态码。

最后通过 Util 去获取响应中的 first_name 信息，做为核心验证点。

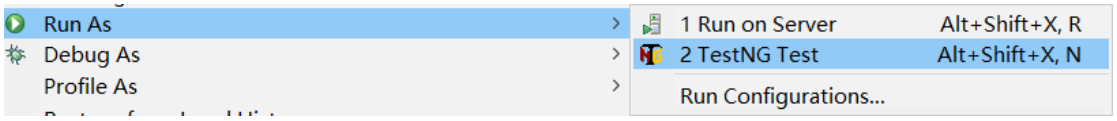
在 getTest 方法里我们使用 Assert.assertEquals 方法做了两个断言：

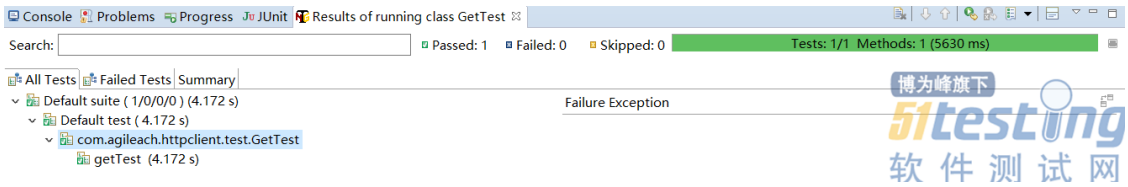
1、判断状态码是否正确；

2、判断 first_name 是否正确。

7.1.2 执行测试并查看结果

可以选中该类，右键点击 Run As 中的 TestNG Test 开始测试，结果如下所示：





第八章、Post，Put 和 Delete 方法

8.1、发送 POST 方法

post 方法和 get 方法是我们在做接口测试时，用的最多的两个请求方法。

在发送请求时它们显著的一个差别就在于，get 方法我们只需要将 url 发送即可，post 我们还需发送一个请求主体；在作用方面，get 方法用于查询，不会改变服务器中的信息；而 Post 方法可用于修改服务器中的信息。

8.1.1 修改 HttpClientUtil 实现发送 POST 请求和获取响应

根据请求的数据是 JSON 对象还是表单形式的键值对，分成了 senPostByJson 和 sendPostByForm 两类方法，其中 sendPostByForm 方法用 ArrayList 存储 NameValuePair 键值对，再设置请求的主体；senPostByJson 方法将对象序列化为 JONS 字符串，再设置请求的主体。

```
public JSONObject sendPostByJson(String url, Object object, HashMap<String, String> headers)
throws Exception {
        httpClient = HttpClients.createDefault();
        HttpPost httpPost = new HttpPost(url);
        try {
            httpPost.setConfig(requestConfig);
            String json = JSON.toJSONString(object);
            StringEntity entity = new StringEntity(json, "utf-8");
            entity.setContentType("application/json");
            httpPost.setEntity(entity);
            if (headers != null) {
                // 加载请求头到 httppost 对象
                for (Map.Entry<String, String> entry : headers.entrySet()) {
                    httpPost.setHeader(entry.getKey(), entry.getValue());
                }
            }
            response = httpClient.execute(httpPost);
            HttpEntity responseEntity = response.getEntity();
            String result = null;
            if (responseEntity != null) {
                result = EntityUtils.toString(responseEntity, "UTF-8");
            }
            EntityUtils.consume(responseEntity);
            JSONObject jsonobj = JSON.parseObject(result);
            jsonobj.put(HTTPSTATUS, response.getStatusLine().getStatusCode());
            return jsonobj;
        } finally {
            httpClient.close();
            response.close();
        }
    }

    public JSONObject sendPostByForm(String url, Map<String, String> form, HashMap<String, String>
headers)
            throws Exception {
```

```
        httpClient = HttpClients.createDefault();
        HttpPost httpPost = new HttpPost(url);
        try {
            httpPost.setConfig(requestConfig);
            // 设置请求主体格式
            if (form.size() > 0) {
                ArrayList<BasicNameValuePair> list = new ArrayList<>();
                form.forEach((key, value) -> list.add(new BasicNameValuePair(key, value)));
                UrlEncodedFormEntity entity = new UrlEncodedFormEntity(list, "UTF-8");
                entity.setContentType("application/x-www-form-urlencoded");
                httpPost.setEntity(entity);
            }
            if (headers != null) {
                // 设置头部信息
                Set<String> set = headers.keySet();
                for (Iterator<String> iterator = set.iterator(); iterator.hasNext();) {
                    String key = iterator.next();
                    String value = headers.get(key);
                    httpPost.setHeader(key, value);
                }
            }
            response = httpClient.execute(httpPost);
            HttpEntity entity = response.getEntity();
            String result = null;
            if (entity != null) {
                result = EntityUtils.toString(entity, "utf-8");
            }
            EntityUtils.consume(entity);
            JSONObject jsonobj = JSON.parseObject(result);
            jsonobj.put(HTTPSTATUS, response.getStatusLine().getStatusCode());
            return jsonobj;
        } finally {
            httpClient.close();
            response.close();
        }
    }

    public JSONObject sendPostByForm(String url, Map<String, String> form) throws Exception {
        return sendPostByForm(url, form, null);
    }
    public JSONObject sendPostByJson(String url, Object object) throws Exception {
        return sendPostByJson(url, object, null);
```

```
    }
```

8.2、发送 Put 方法请求和获取响应

Put 方法请求基本上和 sendPostByForm 方法内容一致，只是将 HttpPost 类换做了 HttpPut 类。

```
public JSONObject sendPut(String url, String entityString, HashMap<String, String> headers)
        throws ClientProtocolException, IOException {
    httpClient = HttpClients.createDefault();
    HttpPut httpPut = new HttpPut(url);
    try {
        httpPut.setConfig(requestConfig);
        httpPut.setEntity(new StringEntity(entityString, "utf-8"));
        if (headers != null) {
            for (Map.Entry<String, String> entry : headers.entrySet()) {
                httpPut.setHeader(entry.getKey(), entry.getValue());
            }
        }
        // 发送 put 请求
        response = httpClient.execute(httpPut);
        HttpEntity entity = response.getEntity();
        String result = null;
        if (entity != null) {
            result = EntityUtils.toString(entity, "utf-8");
        }
        EntityUtils.consume(entity);
        JSONObject jsonobj = JSON.parseObject(result);
        jsonobj.put(HTTPSTATUS, response.getStatusLine().getStatusCode());
        return jsonobj;
    } finally {
        httpClient.close();
        response.close();
    }
}
```

8.3、发送 Delete 方法

Delete 方法更加简单了，它用于删除服务器上的资源，所以执行后只需要返回响应状态码即可。

```
public int sendDelete(String url) throws ClientProtocolException, IOException {
    httpClient = HttpClients.createDefault();
    HttpDelete httpDel = new HttpDelete(url);
```

```
        try {
            httpDel.setConfig(requestConfig);
            // 发送 delete 请求
            response = httpClient.execute(httpDel);
            int statusCode = response.getStatusLine().getStatusCode();
            return statusCode;
        } finally {
            httpClient.close();
            response.close();
        }
}
```

8.4、相关源代码

8.4.1 用于给 sendPostByJson 方法传递对象的 User 类:

```
package com.mytest.httpclient;
public class Uses {
  private String name;
  private String job;
  public User() {
        super();
  }
  public User(String name, String job) {
        super();
        this.name = name;
        this.job = job;
  }
  public String getName() {
        return name;
  }
  public void setName(String name) {
        this.name = name;
  }
  public String getJob() {
        return job;
  }
  public void setJob(String job) {
        this.job = job;
  }
}
```

8.4.2 TestNG 测试类源代码:

```
package com.mytest.httpclient.test;
```

```
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import org.apache.http.HttpEntity;
import org.apache.http.HttpStatus;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.impl.client.CloseableHttpClient;
import org.testng.Assert;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
import com.mytest.httpclient.HttpClientUtil;
import com.mytest.httpclient.Users;
import com.mytest.httpclient.Util;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
public class RequestMethodTest {
  CloseableHttpClient client;
  CloseableHttpResponse closeableHttpResponse;
  HttpEntity responseBody;
  int responseCode;
  HttpClientUtil hcu;
  String url;
  @BeforeTest
  public void setUp() {
        hcu = new HttpClientUtil ();
  }
  @Test
  public void postByFormTest() throws Exception {
        url = "https://reqres.in/api/login";
        Map<String,String> params = new HashMap<String,String>();
        params.put("email", "eve.holt@reqres.in");
        params.put("password", "cityslicka");
        JSONObject responseJson = hcu.sendPostByForm(url, params);
        // 验证状态码是不是 200
        int statusCode = responseJson.getInteger(hcu.HTTPSTATUS);
        Assert.assertEquals(statusCode, HttpStatus.SC_OK, "status code is not 201");
        // 断言响应 json 内容中 name 和 job 是不是期待结果
        // System.out.println(responseString);
        String token = Util.getValueByJPath(responseJson, "token");
        Assert.assertEquals(token, "QpwL5tke4Pnpja7X4", "token is not right");
  }
```

```
    @Test
    public void postByJsonTest() throws Exception {
        // 对象转换成 Json 字符串
        User user = new User("Anthony", "tester");
        url = "https://reqres.in/api/users";
        JSONObject responseJson = hcu.sendPostByJson(url, user);
        // 验证状态码是不是 201
        int statusCode = responseJson.getInteger(hcu.HTTPSTATUS);
        Assert.assertEquals(statusCode, HttpStatus.SC_CREATED, "status code is not 201");
        // 断言响应 json 内容中 name 和 job 是不是期待结果
        // System.out.println(responseString);
        String name = Util.getValueByJPath(responseJson, "name");
        String job = Util.getValueByJPath(responseJson, "job");
        Assert.assertEquals(name, "Anthony", "name is not same");
        Assert.assertEquals(job, "tester", "job is not same");
    }
    @Test
    public void putTest() throws ClientProtocolException, IOException {
        url = "https://reqres.in/api/users/2";
        HashMap<String, String> headermap = new HashMap<String, String>();
        headermap.put("Content-Type", "application/json"); // 这个在 postman 中可以查询到
        // 对象转换成 Json 字符串
        Users user = new Users("Anthony", "automation tester");
        String userJsonString = JSON.toJSONString(user);
        // System.out.println(userJsonString);
        JSONObject responseJson = hcu.sendPut(url, userJsonString, headermap);
        // 验证状态码是不是 200
        int statusCode = responseJson.getInteger(hcu.HTTPSTATUS);
        Assert.assertEquals(statusCode, HttpStatus.SC_OK, "response status code is not 200");
        // 验证名称为 Anthony 的 jon 是不是 automation tester
        String jobString = Util.getValueByJPath(responseJson, "job");
        Assert.assertEquals(jobString, "automation tester", "job is not same");
    }
    @Test
    public void deleteApiTest() throws ClientProtocolException, IOException {
        url = "https://reqres.in/api/users/2";
        int statusCode = hcu.sendDelete(url);
        Assert.assertEquals(statusCode, HttpStatus.SC_NO_CONTENT, "status code is not 204");
    }
}
```

第九章、使用 Extent Reporters 美化测试报告

TestNG 自己生成的测试报告不够美观，我们可以使用 Extent Reporters 来美化测试报告。

9.1、在 pom.xml 中加入支持 extent reporters 的 XML 片段

```
<dependency>
    <groupId>com.aventstack</groupId>
    <artifactId>extentreports</artifactId>
    <version>3.1.5</version>
</dependency>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.0.0-M4</version>
            <configuration>
                <suiteXmlFiles>
                    <!--此处testng.xml即为要运行的testng.xml文件 -->
                    <suiteXmlFile>testng.xml</suiteXmlFile>
                </suiteXmlFiles>
            </configuration>
        </plugin>
    </plugins>
</build>
```

增加内容后的完整的 pom.xml 文件内容如下:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mytest</groupId>
  <artifactId>apidemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
      <dependency>
          <groupId>org.testng</groupId>
          <artifactId>testng</artifactId>
          <version>7.1.0</version>
      </dependency>
      <dependency>
          <groupId>org.apache.httpcomponents</groupId>
          <artifactId>httpclient</artifactId>
```

```
            <version>4.5.11</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.62</version>
        </dependency>
        <dependency>
            <groupId>com.aventstack</groupId>
            <artifactId>extentreports</artifactId>
            <version>3.1.5</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M4</version>
                <configuration>
                    <suiteXmlFiles>
                        <!--此处 testng.xml 即为要运行的 testng.xml 文件 -->
                        <suiteXmlFile>testng.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

9.2、新建一个实现 IReporter 接口的类

在项目目录 src/main/java 下的包 com.mytest.httpclient 下新建一个实现了 IReporter 接口的类，类名为 ExtentTestNGReporterListener，其代码如下(代码非本人原创，来源于互联网，但证实正确有用):

```
package com.mytest.httpclient;
import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.ResourceCDN;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.model.TestAttribute;
```

```
import com.aventstack.extentreports.reporter.ExtentHtmlReporter;
import com.aventstack.extentreports.reporter.configuration.ChartLocation;
import com.aventstack.extentreports.reporter.configuration.Theme;
import org.testng.*;
import org.testng.xml.XmlSuite;
import java.io.File;
import java.util.*;
public class ExtentTestNGReporterListener implements IReporter {
    //生成的路径以及文件名
    private static final String OUTPUT_FOLDER = "test-output/";
    private static final String FILE_NAME = "index.html";
    private ExtentReports extent;
    public void generateReport(List<XmlSuite> xmlSuites, List<ISuite> suites, String
outputDirectory) {
        init();
        boolean createSuiteNode = false;
        if(suites.size()>1){
            createSuiteNode=true;
        }
        for (ISuite suite : suites) {
            Map<String, ISuiteResult> result = suite.getResults();
            //如果 suite 里面没有任何用例，直接跳过，不在报告里生成
            if(result.size()==0){
                continue;
            }
            //统计 suite 下的成功、失败、跳过的总用例数
            int suiteFailSize=0;
            int suitePassSize=0;
            int suiteSkipSize=0;
            ExtentTest suiteTest=null;
            //存在多个 suite 的情况下，在报告中将同一个 suite 的测试结果归为一类，创建一
级节点。
            if(createSuiteNode){
                suiteTest = extent.createTest(suite.getName()).assignCategory(suite.getName());
            }
            boolean createSuiteResultNode = false;
            if(result.size()>1){
                createSuiteResultNode=true;
            }
            for (ISuiteResult r : result.values()) {
                ExtentTest resultNode;
                ITestContext context = r.getTestContext();
```

```
                    if(createSuiteResultNode){
                        //没有创建 suite 的情况下，将在 SuiteResult 的创建为一级节点，否则创
建为 suite 的一个子节点。
                        if( null == suiteTest){
                            resultNode = extent.createTest(r.getTestContext().getName());
                        }else{
                            resultNode = suiteTest.createNode(r.getTestContext().getName());
                        }
                    }else{
                        resultNode = suiteTest;
                    }
                    if(resultNode != null){
                        resultNode.getModel().setName(suite.getName()+" :
"+r.getTestContext().getName());
                        if(resultNode.getModel().hasCategory()){
                            resultNode.assignCategory(r.getTestContext().getName());
                        }else{

resultNode.assignCategory(suite.getName(),r.getTestContext().getName());
                        }
                        resultNode.getModel().setStartTime(r.getTestContext().getStartDate());
                        resultNode.getModel().setEndTime(r.getTestContext().getEndDate());
                        //统计 SuiteResult 下的数据
                        int passSize = r.getTestContext().getPassedTests().size();
                        int failSize = r.getTestContext().getFailedTests().size();
                        int skipSize = r.getTestContext().getSkippedTests().size();
                        suitePassSize += passSize;
                        suiteFailSize += failSize;
                        suiteSkipSize += skipSize;
                        if(failSize>0){
                            resultNode.getModel().setStatus(Status.FAIL);
                        }
                        resultNode.getModel().setDescription(String.format("Pass: %s ; Fail: %s ;
Skip: %s ;",passSize,failSize,skipSize));
                    }
                    buildTestNodes(resultNode,context.getFailedTests(), Status.FAIL);
                    buildTestNodes(resultNode,context.getSkippedTests(), Status.SKIP);
                    buildTestNodes(resultNode,context.getPassedTests(), Status.PASS);
                }
                if(suiteTest!= null){
                    suiteTest.getModel().setDescription(String.format("Pass: %s ; Fail: %s ;
Skip: %s ;",suitePassSize,suiteFailSize,suiteSkipSize));
                    if(suiteFailSize>0){
```

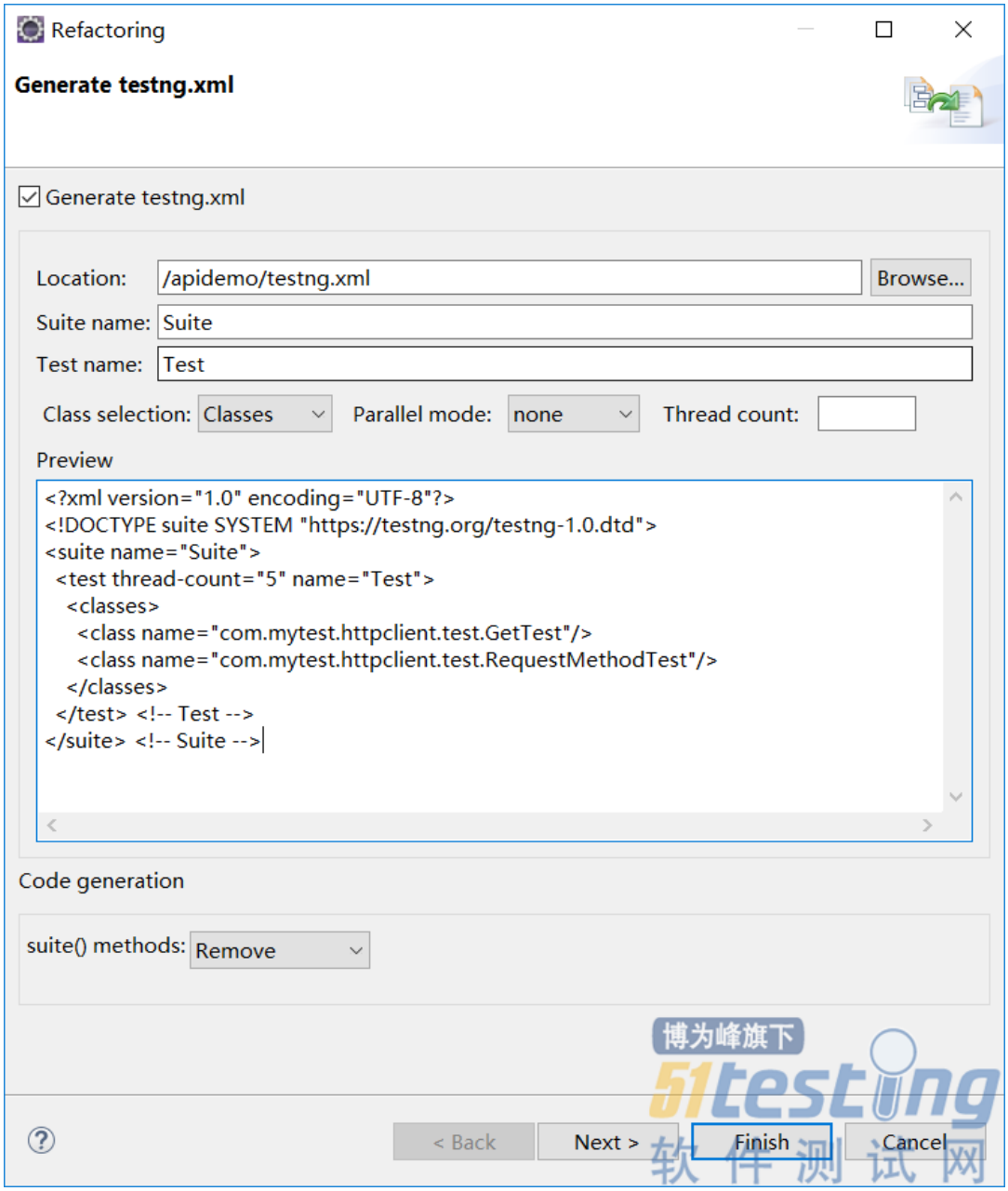
```
                    suiteTest.getModel().setStatus(Status.FAIL);
                }
            }

        }
        extent.flush();
    }
    private void init() {
        //文件夹不存在的话进行创建
        File reportDir= new File(OUTPUT_FOLDER);
        if(!reportDir.exists()&& !reportDir .isDirectory()){
            reportDir.mkdir();
        }
        ExtentHtmlReporter htmlReporter = new ExtentHtmlReporter(OUTPUT_FOLDER +
FILE_NAME);
        //设置静态文件的 DNS
        //解决 cdn.rawgit.com 访问不了的情况
        htmlReporter.config().setResourceCDN(ResourceCDN.EXTENTREPORTS);
        htmlReporter.config().setDocumentTitle("自动化接口测试报告");
        htmlReporter.config().setReportName("自动化接口测试报告");
        htmlReporter.config().setChartVisibilityOnOpen(true);
        htmlReporter.config().setTestViewChartLocation(ChartLocation.TOP);
        htmlReporter.config().setTheme(Theme.STANDARD);
        htmlReporter.config().setCSS(".node.level-1    ul{ display:none;} .node.level-1.active
ul{display:block;}");
        extent = new ExtentReports();
        extent.attachReporter(htmlReporter);
        extent.setReportUsesManualConfiguration(true);
    }
    private void buildTestNodes(ExtentTest extenttest, IResultMap tests, Status status) {
        //存在父节点时，获取父节点的标签
        String[] categories=new String[0];
        if(extenttest != null ){
            List<TestAttribute> categoryList = extenttest.getModel().getCategoryContext().getAll();
            categories = new String[categoryList.size()];
            for(int index=0;index<categoryList.size();index++){
                categories[index] = categoryList.get(index).getName();
            }
        }
        ExtentTest test;
        if (tests.size() > 0) {
            //调整用例排序，按时间排序
```

```
Set<ITestResult> treeSet = new TreeSet<ITestResult>(new Comparator<ITestResult>() {
    public int compare(ITestResult o1, ITestResult o2) {
        return o1.getStartMillis()<o2.getStartMillis()?-1:1;
    }
});
treeSet.addAll(tests.getAllResults());
for (ITestResult result : treeSet) {
    Object[] parameters = result.getParameters();
    String name="";
    //如果有参数，则使用参数的 toString 组合代替报告中的 name
    for(Object param:parameters){
        name+=param.toString();
    }
    if(name.length()>0){
        if(name.length()>50){
            name= name.substring(0,49)+"...";
        }
    }else{
        name = result.getMethod().getMethodName();
    }
    if(extenttest==null){
        test = extent.createTest(name);
    }else{
        //作为子节点进行创建时，设置同父节点的标签一致，便于报告检索。
        test = extenttest.createNode(name).assignCategory(categories);
    }
    for (String group : result.getMethod().getGroups())
        test.assignCategory(group);

    List<String> outputList = Reporter.getOutput(result);
    for(String output:outputList){
        //将用例的 log 输出报告中
        test.debug(output);
    }
    if (result.getThrowable() != null) {
        test.log(status, result.getThrowable());
    }
    else {
        test.log(status, "Test " + status.toString().toLowerCase() + "ed");
    }                    test.getModel().setStartTime(getTime(result.getStartMillis()));
    test.getModel().setEndTime(getTime(result.getEndMillis()));
}
```

```
            }
        }
        private Date getTime(long millis) {
            Calendar calendar = Calendar.getInstance();
            calendar.setTimeInMillis(millis);
            return calendar.getTime();
        }
    }
```

9.3、创建 testng.xml 文件并添加相应的监听器

右键点击项目名，在弹出的菜单中选择 TestNG → Convert to TestNG,在弹出的窗口中点击 Finish 按钮,这样能够自动生成项目的 testng.xml 文件。



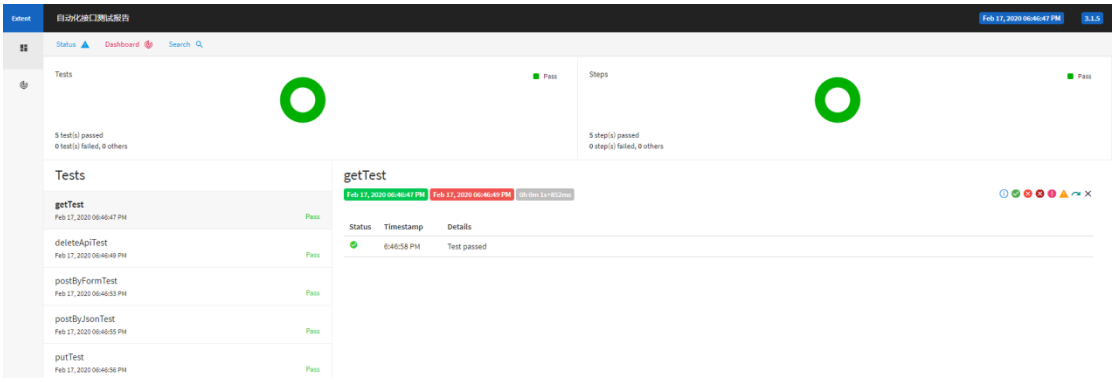
在 testng.xml 中把刚才创建的类 ExtentTestNGReporterListener 加入到监听器中，这样在测试运行时这个监听器才会起作用。最后完成的 testng.xml 文件内容如下:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
    <listeners>
      <listener class-name="com.mytest.httpclient.ExtentTestNGReporterListener" />
  </listeners>
   <test thread-count="5" name="Test">
      <classes>
          <class name="com.mytest.httpclient.test.GetTest"/>
<class name="com.mytest.httpclient.test.RequestMethodTest"/>
      </classes>
   </test> <!-- Test -->
</suite> <!-- Suite -->
```

9.4、使用 Maven 执行测试

最后，我们用 Maven 来执行下测试，这样可以自动生成 extent reporters 的测试报告。

右键点击项目名，点击 Run As →7 Maven test，这样会以 Maven 的方式自动执行测试并生成相应的测试报告，执行完毕后，我们点击 test-output 文件夹中的 index.html,就会看到生成的 extent reporters 格式的测试报告。



第十章、使用 Jenkins 构建自动化测试持续集成

现在代码可以运行了，但是每次运行都需要我们手工去执行，并且测试报告也只能在执行测试的电脑上才能看到，我们希望能够定时自动执行测试，并且能够做到自动发送测试报告到相关人员的电子邮箱中。

Jenkins 正好可以很好的完成以上诉求，那我们首先下载并安装好 Jenkins(网上有很多相关资料和教程，这里略过)。

接下来说下具体的配置步骤:

10.1、相关的工具软件在 Jenkins 服务器中安装和配置

JDK: 必须配置，Java 写的程序哦!

Maven: 必须配置，我们建立的就是 Maven 类型的项目。

10.2、相关插件下载

Maven Integration: 必须，我们需要在 Jenkins 中建立一个 Maven 项目;

HTML Publisher plugin: 必须，extent reporters 美化报告替换 testng 本来的报告就是为了美观，要在 Jenkins 中展示必须安装此插件。

Groovy: 必须，Jenkins 不支持异类样式 CSS，所以 Groovy 插件是为了解决 HTML Publisher plugin 在展示 extent reporets 时能够正确的显示。

10.3、相关工具软件和插件在 Jenkins 管理界面中配置

以管理员登录 Jenkins 成功，先后点击左侧菜单中的 Manage Jenkins → Global Tool Configuration:

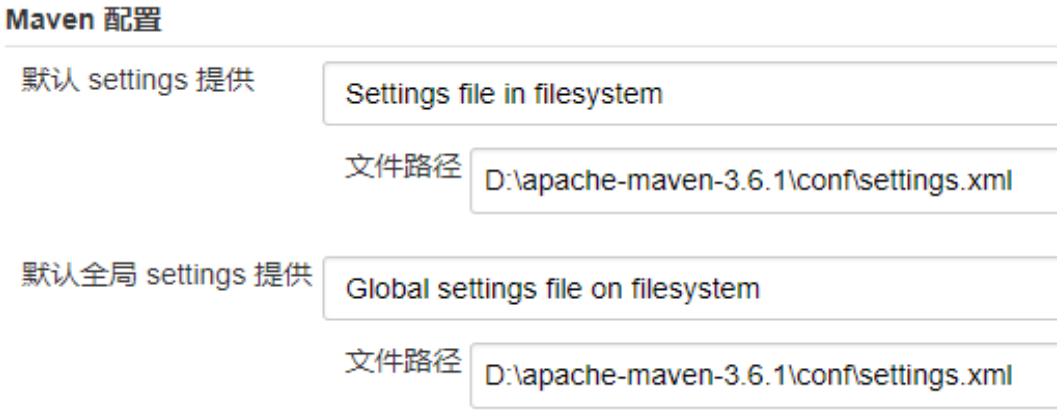
10.3.1 JDK

别名填写一个容易辨识的就可以，JAVA_HOME 中填写该环境变量的值。

10.3.2 Maven

Maven 的名字填写一个容易辨识的就可以，MAVE_HOME 中填写该环境变量的值。

10.3.3 Maven 配置

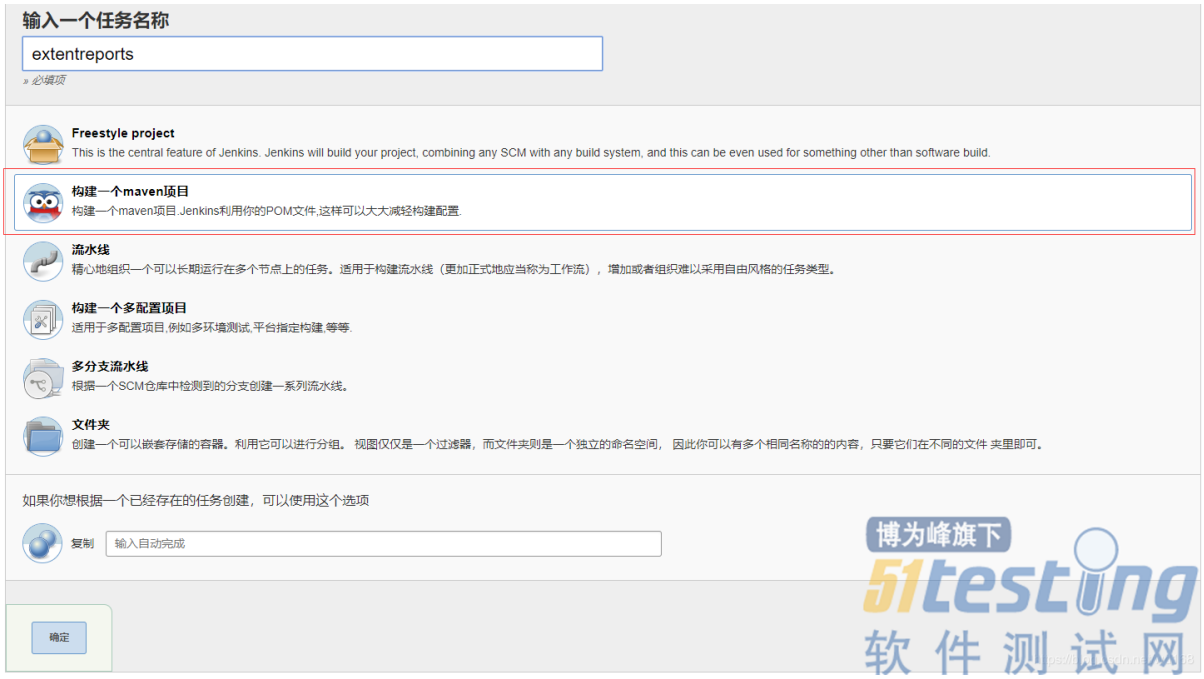


Maven 配置需要填写两个 settings 的文件路径，填写实际配置的路径即可。

10.3.4 Groovy

这里选中 Install automatically，就不用自己再去下载安装了，现在 Jenkins 功能越来越强大了。版本选择一个最新的就可以了。

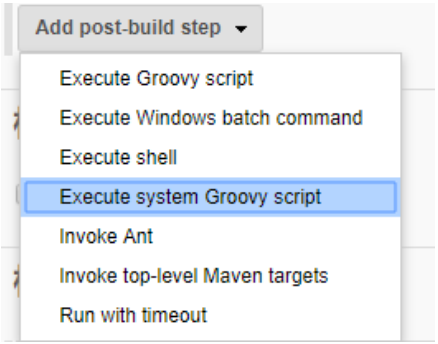
10.4、新建一个 Maven 类型的项目



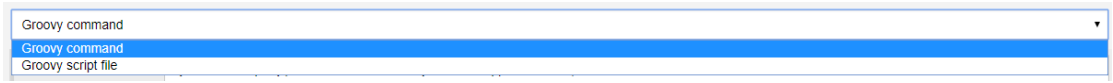
在 Build 中正确配置好 Maven 设置:

这里的 Root POM 中填写的路径是相对于${workspace},也可以写成绝对路径:${workspace}\apidemo\pom.xml，这里 apidemo 是对应的 exclipse 项目文件夹，需要改为自己的项目文件夹名称。

接下来在 Post Steps 中选择构建步骤“Execute system Groovy script” ，



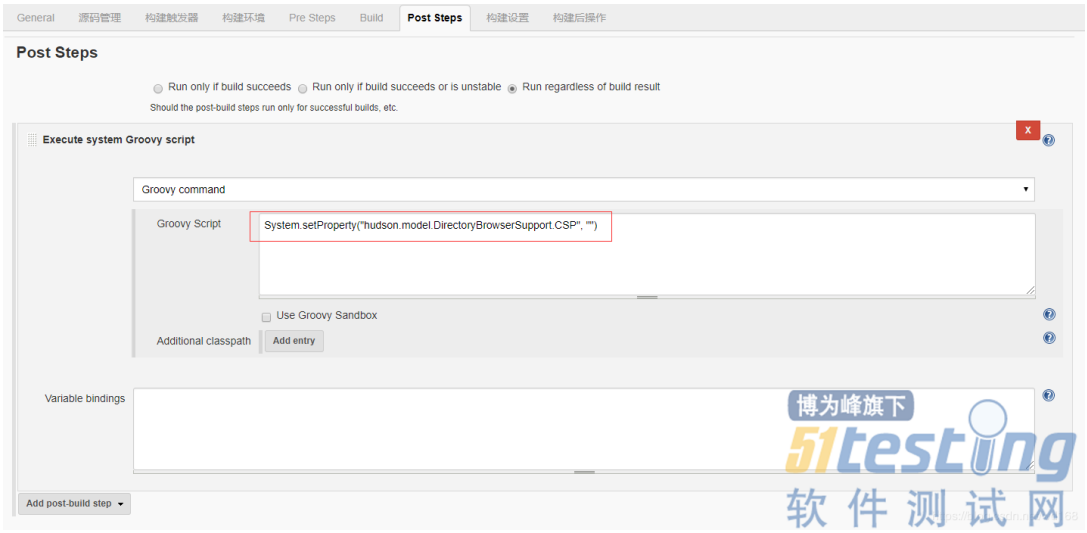
选择 Groovy command，



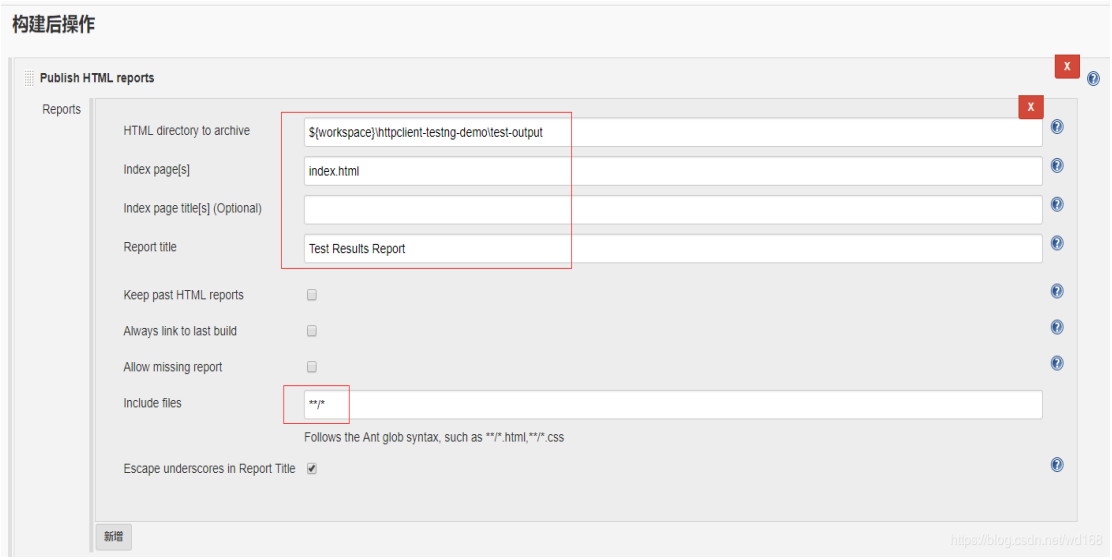
填入

System.setProperty("hudson.model.DirectoryBrowserSupport.CSP", ""),

如下图所示：



在构建后设置中选择 Publishe HTML reports，在 HTML directory to archive 中填写生成的测试报告所在的文件夹，这里需要填写的就是在 ExtentTestNGReporterListener.java 文件中定义的文件夹路径，Index page[s]填写的也是 ExtentTestNGReporterListener.java 文件中定义的文件路径，Report title 中填写的是报告标题，Include files 保持默认即可。


构建后操作
Publish HTML reports
Reports
HTML directory to archive
${workspace}\httpclient-testng-demo\test-output
Index page[s]
index.html
Index page title[s] (Optional)
Report title
Test Results Report
Keep past HTML reports
Always link to last build
Allow missing report
Include files
**/*
Follows the Ant glob syntax, such as **/*.html,**/*.css
Escape underscores in Report Title
新增

Django 环境搭建及 Restful 接口测试实践

◆ 作者：桃 子

本文分为两部分，一是搭建 Django 接口开发环境，二是接口测试，接口测试分别介绍了使用 postman 工具测试、Request+Unittest 测试和 Django 自带测试模块

一、 Django 接口开发

（1）Django 安装

输入如下命令即可安装 Django，注意需要提前配置好 Python 环境，这里选择的 dja

```
pip install django
```

安装校验：在 Windows 命令提示符下输入 `django-admin` 命令回车。

提示如下内容则说明安装成功


```
C:\Users\Shuqing>django-admin
```

实践：输入 Django-admin.py 运行成功

（2）Django rest framework 安装

安装

```
pip install djangorestframework #Django REST Framework
pip install markdown       # Markdown support for the browsable API.
pip install django-filter  # Filtering support
```

（3）项目创建

d 盘创建新项目

命令：Django-admin startproject Django_restful

Django_restful 项目中创建 api 应用

命令：python manage.py startapp api

```
D:\>django-admin startproject django_restful

D:\>cd django_restful

D:\django_restful>python manage.py startapp api

D:\django_restful>
```

进入 D:\Django_restful\Django_restful 目录下，打开 setting.py 文件创建 api 和 rest_framework

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'api',
```

配置 rest_framewor 权限

在 setting.py 文件末尾添加权限代码:

```
REST_FRAMEWORK = {
  DEFAULT_PERMISSION_CLASSES': [
   'rest_framework.permissions.IsAuthenticated',
  ]
}
```

（4）数据库迁移

cmd 下输入命令： python manage.py migrate

提示如下图迁移成功

```
D:\django_restful>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying sessions.0001_initial... OK
```

（5）创建超级管理员

cmd 下输入命令： python manage.py createsuperuser

依次输入账号、邮箱、密码，如下图

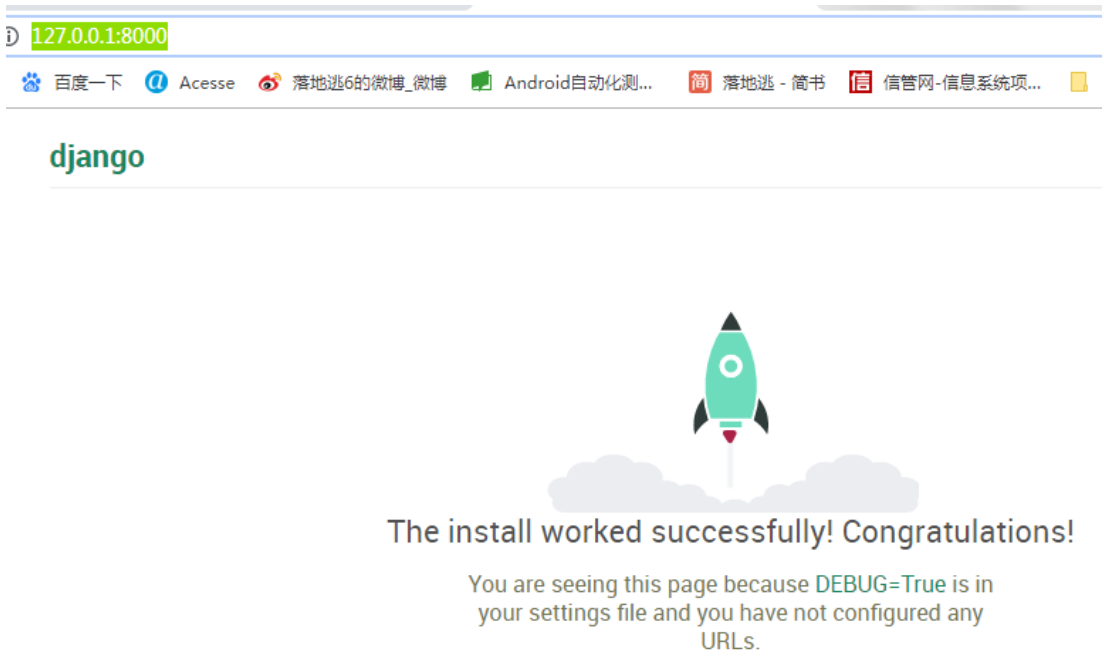
```
D:\>cd django_restful

D:\django_restful>python manage.py createsuperuser
Username (leave blank to use 'administrator'): wx
Email address: 635907412@qq.com
Password:
Password (again):
Superuser created successfully.
```

（6）启动 server

启动 Django ，登录创建的用户 python manage.py runserver

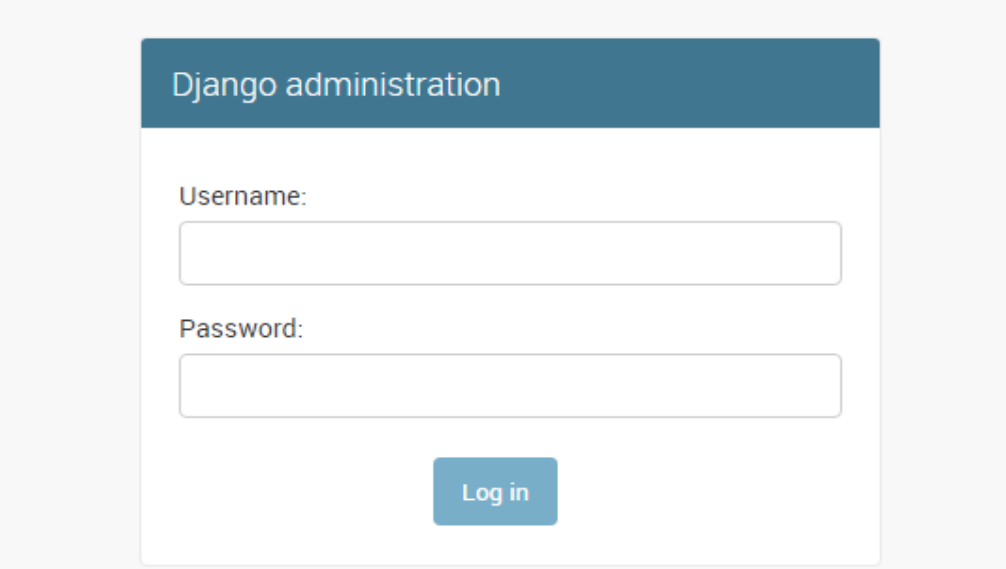
```
D:\django_restful>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
March 13, 2020 - 17:11:52
Django version 2.0.13, using settings 'django_restful.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

浏览器输入地址：http://127.0.0.1:8000/

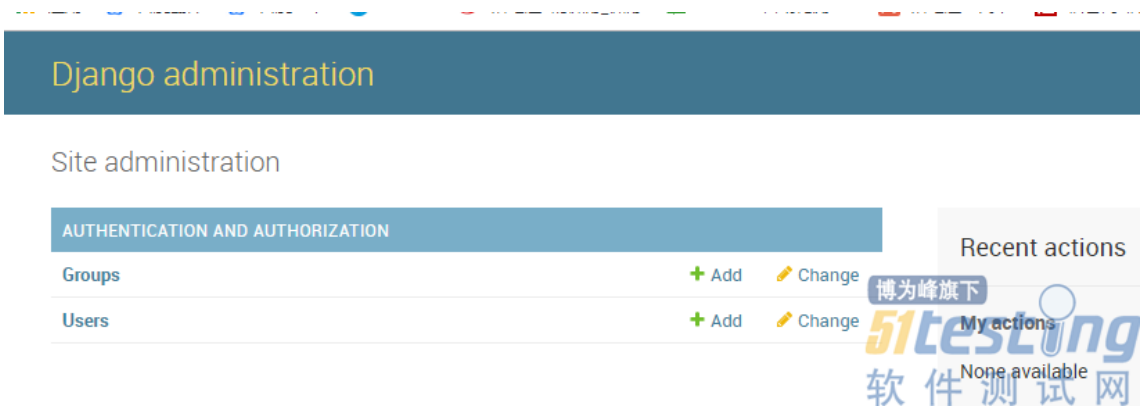


（7）登录超级管理员账户

浏览器输入地址：http://127.0.0.1:8000/admin

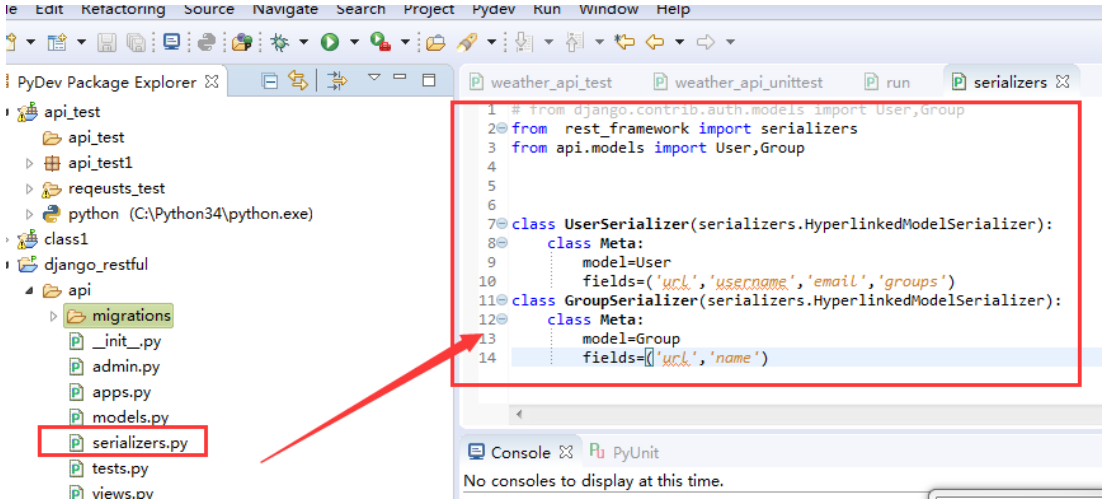


输入账号、密码后登录



（8）数据序列化

在 API 文件夹下新建 serializers.py 文件，定义 API 返回形式，返回哪些字段，返回怎样的格式等



代码：

```
# from Django.contrib.auth.models import User,Group
from   rest_framework import serializers
from api.models import User,Group

class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model=User
        fields=('url','username','email','groups')
class GroupSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model=Group
                fields=('url','name')
```

（9）创建视图

Django rest framework 使用 ViewSets 定义视图的展现形式，如何向用户展示数据，展示什么数据等

在 Api 下 vies.py 文件编写如下代码

```
from Django.shortcuts import render
# from Django.contrib.auth.models import User,Group
from rest_framework import viewsets
from api.serializers import UserSerializer,GroupSerializer
from api.models import User,Group

# Create your views here.

class UserViewSet(viewsets.ModelViewSet):
    """
        retrieve:
            Return a user instance.
        list:
            Return all users,odered by most recent joined.
        create:
            Create a new user.
        delete:
            Remove a existing user
        partial_update:
            Update one or more fields on a existing user.
        update:
            Update a user.
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer

class GroupViewSet(viewsets.ModelViewSet):
    """
            retrieve:
                Return a group instance.
            list:
                Return all groups, ordered by most recently joined.
            create:
                Create a new group.
            delete:
                Remove an existing group.
            partial_update:
                Update one or more fields on an existing group.
```

```
            update:
                Update a group.
        """
    queryset = Group.objects.all()
serializer_class = GroupSerializer
```

（10）URL 路由配置

打开 Django_restful/urls.py 文件，填写下面的代码配置路由

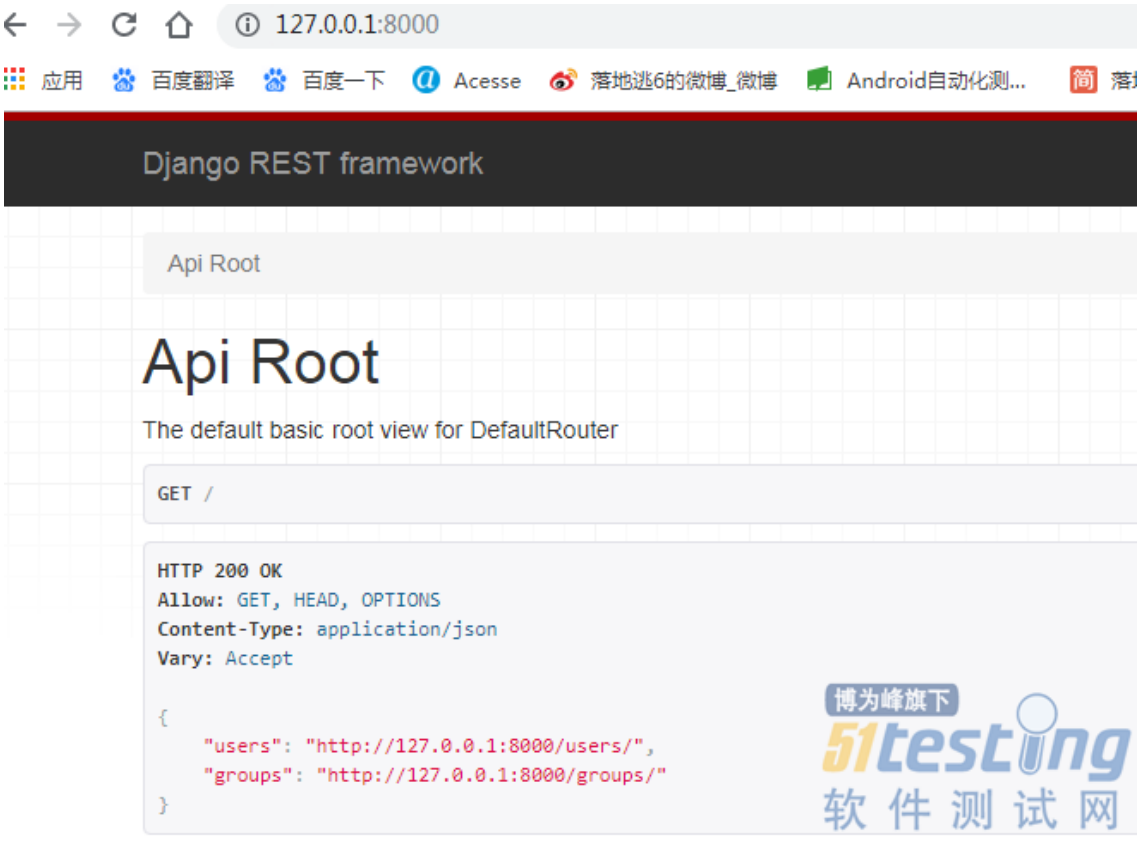
```
"""Django_restful URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.Djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from Django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from Django.contrib import admin
from Django.urls import path
from Django.conf.urls import include
from rest_framework import routers
from api import views
#from rest_framework.schemas import get_schema_view
#from rest_framework_swagger.renderers import SwaggerUIRenderer,OpenAPIRenderer

#schema_view=get_schema_view(title='API',renderer_classes=[OpenAPIRenderer,SwaggerUIRenderer
])

router=routers.DefaultRouter()
router.register(r'users',views.UserViewSet)
router.register(r'groups',views.GroupViewSet)

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include(router.urls)),
```

```
    path('api-auth/',include('rest_framework.urls',namespace='rest_framework')),
    path('docs/',schema_view,name='docs')
]
```

cmd 下重新启动服务：python manage.py runserver

```
# from Django.contrib.auth.models import User,Group
from  rest_framework import serializers
from api.models import User,Group

class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model=User
        fields=('url','username','email','groups')
class GroupSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model=Group
        fields=('url','name')
```

（11）打开 API 主页

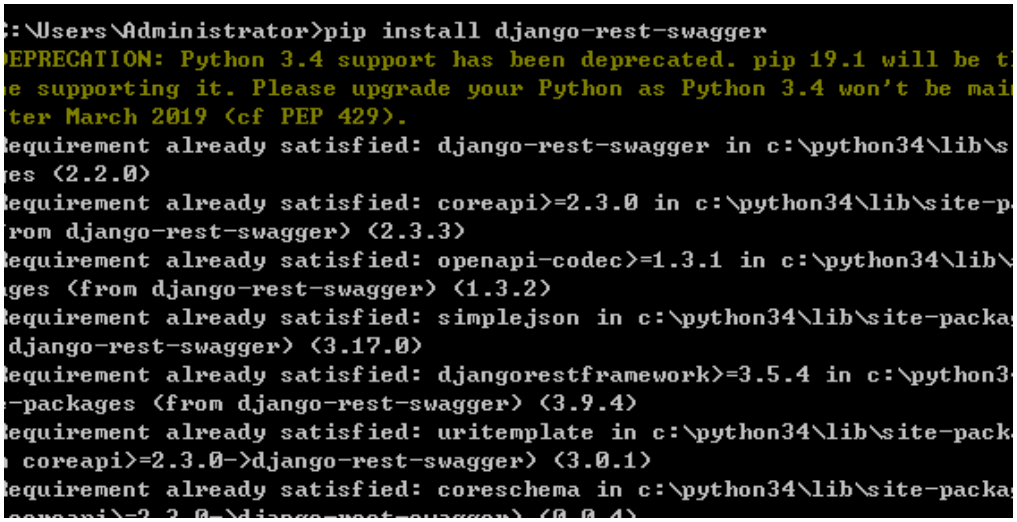


（12）Swagger 接口文档生成

Swagger 是一个框架，是一个 restful 风格的 web 服务，每次接口有变动，接口文档也会自动更新

Django 接入 swagger

cmd 安装 swagger ： pip install Django-rest-swagger



打开 setting.py 文件，添加 Django-rest-swagger 应用

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'api',
    'rest_framework_swagger',

]
```

打开 views.py 填写如下代码:

```
from Django.shortcuts import render
from Django.contrib.auth.models import User,Group
from rest_framework import viewsets
from api.serializers import UserSerializer,GroupSerializer
#from api.models import User,Group

# Create your views here.
class UserViewSet(viewsets.ModelViewSet):
    """
        retrieve:
```

```
            Return a user instance.
        list:
            Return all users,odered by most recent joined.
        create:
            Create a new user.
        delete:
            Remove a existing user
        partial_update:
            Update one or more fields on a existing user.
        update:
            Update a user.
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer

class GroupViewSet(viewsets.ModelViewSet):
    """
            retrieve:
                Return a group instance.
            list:
                Return all groups, ordered by most recently joined.
            create:
                Create a new group.
            delete:
                Remove an existing group.
            partial_update:
                Update one or more fields on an existing group.
            update:
                Update a group.
        """
    queryset = Group.objects.all()
    serializer_class = GroupSerializer
```

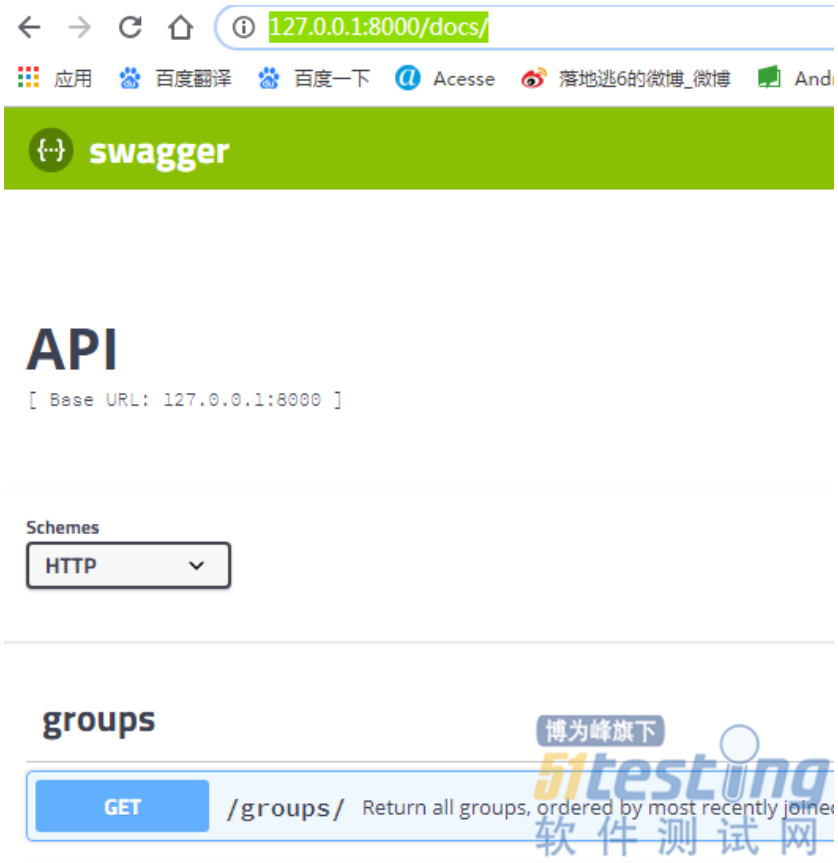
打开 url.py 添加如下代码

```
"""Django_restful URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.Djangoproject.com/en/2.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
```

```
Class-based views
    1. Add an import:   from other_app.views import Home
    2. Add a URL to urlpatterns:   path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from Django.urls import include, path
    2. Add a URL to urlpatterns:   path('blog/', include('blog.urls'))
"""
from Django.contrib import admin
from Django.urls import path
from Django.conf.urls import include
from rest_framework import routers
from api import views
from rest_framework.schemas import get_schema_view
from rest_framework_swagger.renderers import SwaggerUIRenderer,OpenAPIRenderer
schema_view=get_schema_view(title='API',renderer_classes=[OpenAPIRenderer,SwaggerUIRenderer])
router=routers.DefaultRouter()
router.register(r'users',views.UserViewSet)
router.register(r'groups',views.GroupViewSet)
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include(router.urls)),
    path('api-auth/',include('rest_framework.urls',namespace='rest_framework')),
    path('docs/',schema_view,name='docs')
]
```

启动服务，浏览器打开网址 http://127.0.0.1:8000/docs/

界面如下:



二、 Restful 接口测试

可以使用工具测试如 postman 或者 jmeter，也可以通过 request+unittest 脚本测试

测试场景:

正常测试：数据的增删改查

异常测试：未授权，参数异常等

1、postman 测试

（1）user 接口

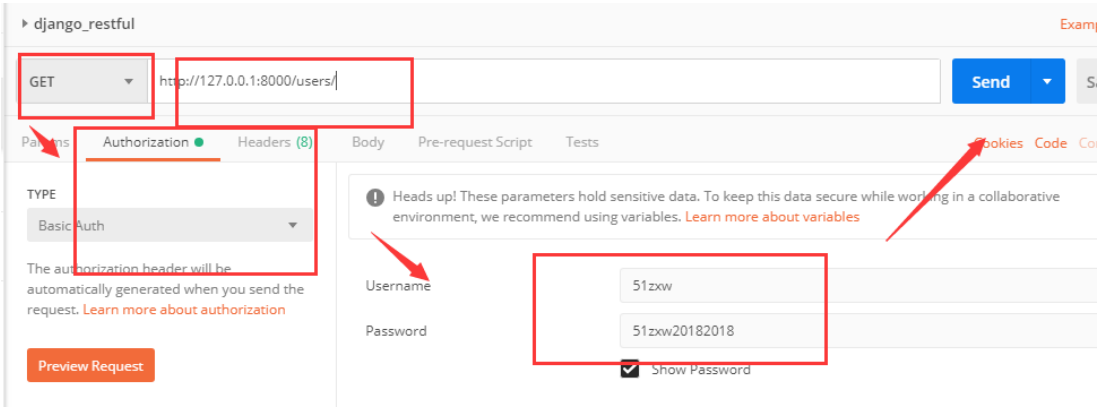
- 查询功能

get 方法

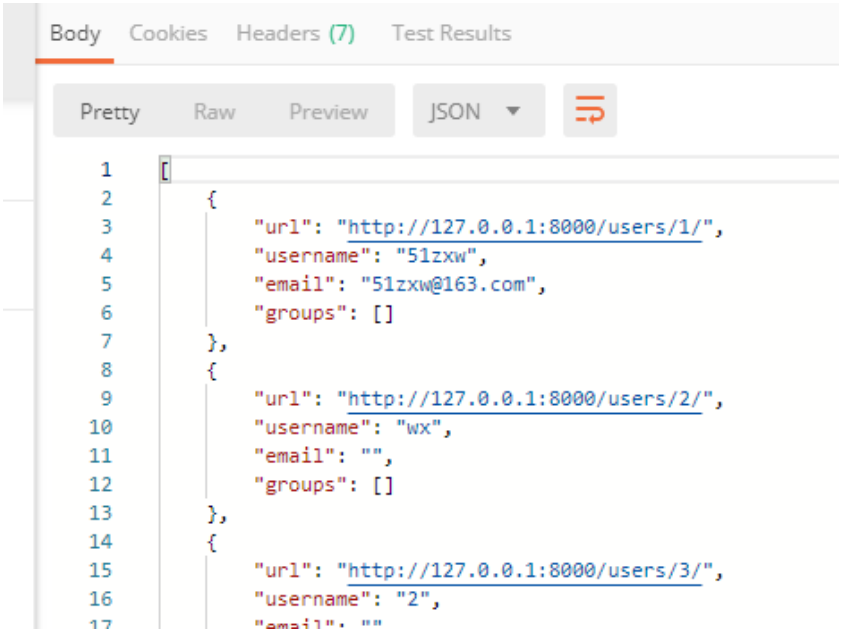
url 输入 http://127.0.0.1:8000/users

授权：authorization 选择 basic auth，输入账户、密码

点击 send



send 结果:



- 创建用户

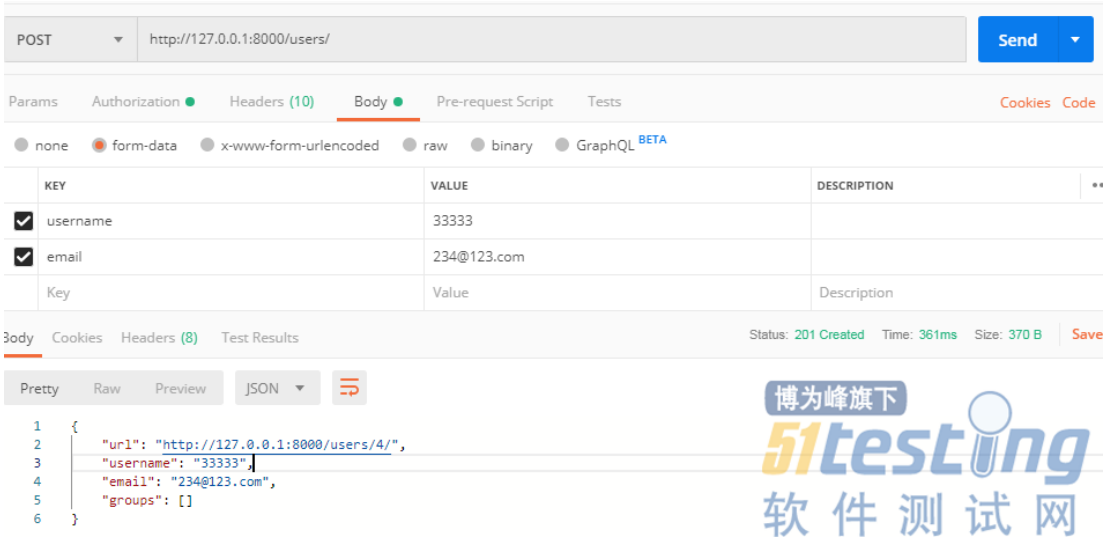
POST 方法

url 输入/http://127.0.0.1:8000/users

授权：authorization 选择 basic auth，输入账户、密码

body 下输入 username 33333

点击 send



- **修改用户**

修改 ueser id 为 2 的用户名改为 lbz

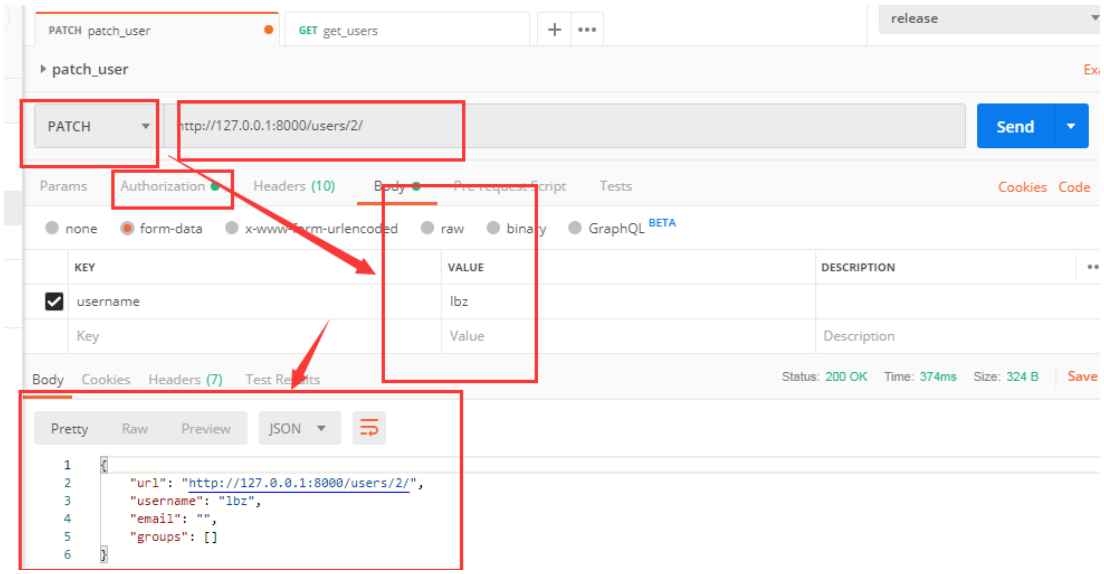
PATCH 方法

url 输入/http://127.0.0.1:8000/users/2/

授权：authorization 选择 basic auth，输入账户、密码

body 下输入 username lbz

点击 send



- **删除测试**

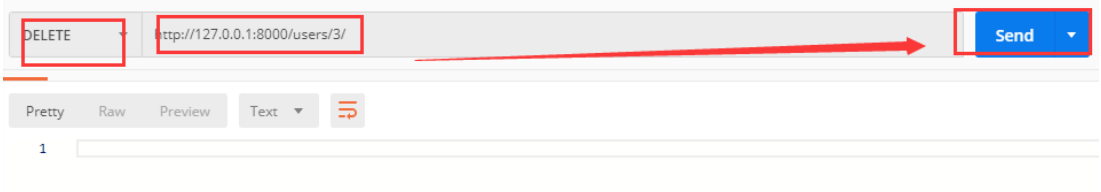
删除 ueser id 为 3 的用户

PATCH 方法

url 输入/http://127.0.0.1:8000/users/3/

授权：authorization 选择 basic auth，输入账户、密码

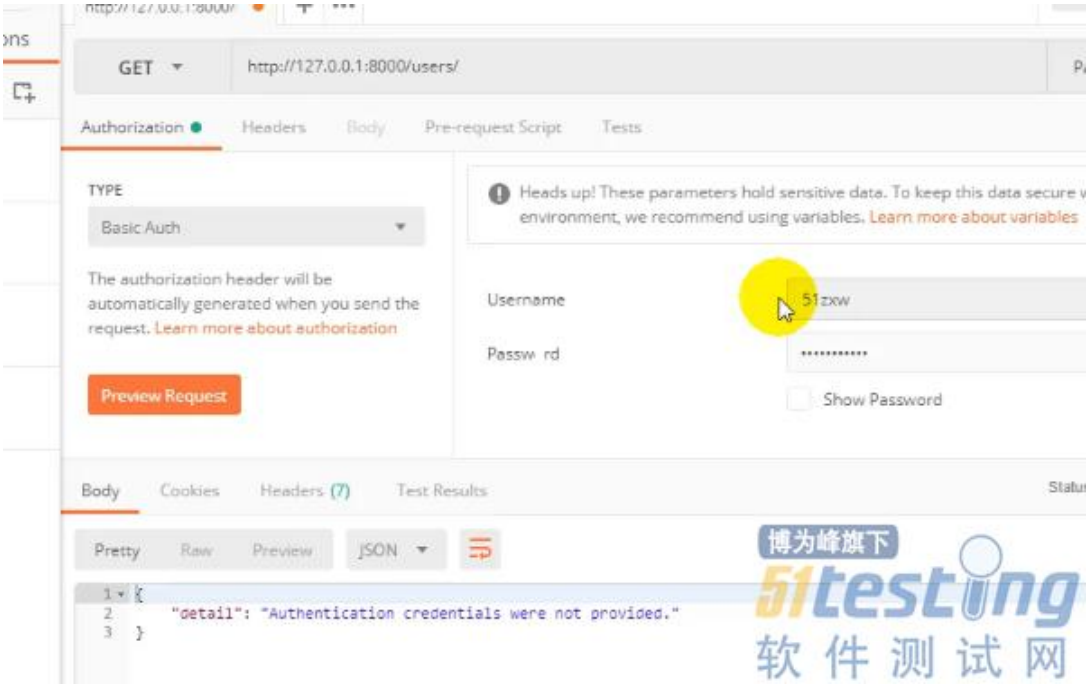
点击 send



- 未授权测试

输入路径如 http://127.0.0.1:8000/users/

不授权，点击 send，提示如下



2、脚本测试

（1）Request+Unittest 测试

```
#-*-coding:GBK -*-
import requests
import unittest
class UserTest(unittest.TestCase):
    def setUp(self):
```

```
        self.base_url='http://127.0.0.1:8000/users'
        self.auth=('51zxw','51zxw20182018')
    #查询功能
    def test_get_user(self):
        r=requests.get(self.base_url+'/1/',auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'51zxw')
        self.assertEqual(result['email'],'51zxw@163.com')
    #添加功能
    def test_add_user(self):
        form_data={'username':'add1','email':'666@162.com','groups':'http://127.0.0.1:8000/groups/2/'}
        r=requests.post(self.base_url+'/',data=form_data,auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'add1')
    #修改功能
    def test_update_user(self):
        form_data={'email':'update@162.com'}
        r=requests.patch(self.base_url+'/2/',data=form_data,auth=self.auth)
        result=r.json()
        self.assertEqual(result['email'],'update@162.com')
    #删除功能
    def test_delete_user(self):
        r=requests.delete(self.base_url+'/3/',auth=self.auth)
        self.assertEqual(r.status_code,404)
    #无授权
    def test_no_auth(self):
        r=requests.get(self.base_url)
        result=r.json()
        self.assertEqual(result['detail'],'Authentication credentials were not provided.')
if __name__=='_main_':
    unittest.main()
```

在 API 目录下新建 test_unittest.py,输入下面代码，实现增删改查功能

（2）Django 自带测试模块

打开 API 目录下新建 tests 文件输入下面代码，实现增删改查功能

```
#-*-coding:GBK -*-
from Django.test import TestCase
import requests
class UserTest(TestCase):
    def setUp(self):
        self.base_url='http://127.0.0.1:8000/users'
```

```
        self.auth=('51zxw','51zxw20182018')
    #查询功能
    def test_get_user(self):
        r=requests.get(self.base_url+'/1/',auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'51zxw')
        self.assertEqual(result['email'],'51zxw@163.com')
    #添加功能
    def test_add_user(self):
        form_data={'username':'add1','email':'666@162.com','groups':'http://127.0.0.1:8000/groups/2/'}
        r=requests.post(self.base_url+'/',data=form_data,auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'add1')
    #修改功能
    def test_update_user(self):
        form_data={'email':'update@162.com'}
        r=requests.patch(self.base_url+'/2/',data=form_data,auth=self.auth)
        result=r.json()
        self.assertEqual(result['email'],'update@162.com')
    #删除功能
    def test_delete_user(self):
        r=requests.delete(self.base_url+'/3/',auth=self.auth)
            self.assertEqual(r.status_code,404)
    #无授权
    def test_no_auth(self):
        r=requests.get(self.base_url)
        result=r.json()
        self.assertEqual(result['detail'],'Authentication credentials were not provided.')
```

运行:

cmd 下: python manage.py test

查看结果:

```
Traceback (most recent call last):
  File "D:\django_restful\api\tests.py", line 25, in test_add_user
    self.assertEqual(result['username'],'add1')
AssertionError: ['A user with that username already exists.'] != 'add1'

----------------------------------------------------------------------
Ran 5 tests in 1.061s

FAILED (failures=1, errors=1)
Destroying test database for alias 'default'...
```

浏览器输入路径查看：http://127.0.0.1:8000/users/


```
[
    {
        "url": "http://127.0.0.1:8000/users/1/",
        "username": "51zxw",
        "email": "51zxw@163.com",
        "groups": []
    },
    {
        "url": "http://127.0.0.1:8000/users/2/",
        "username": "lbz",
        "email": "update@162.com",
        "groups": []
    },
    {
        "url": "http://127.0.0.1:8000/users/5/",
        "username": "add",
        "email": "666@162.com",
        "groups": [
            "http://127.0.0.1:8000/groups/2/"
        ]
    },
    {
        "url": "http://127.0.0.1:8000/users/6/",
        "username": "add1",
        "email": "666@162.com",
        "groups": [
            "http://127.0.0.1:8000/groups/2/"
        ]
    }
]
```

Pytest 自动化测试探索

◆作者：枫　叶

摘要： Pytest 是成熟的功能齐全的 Python 测试工具，有助于编写更好的程序。

一、Pytest 基本知识

参考官方文档翻译过来，了解一下 Pytest 知识点。

1、Pytest 中可以按节点 ID 运行测试。

在命令行中指定测试方法的另一个示例：

```
pytest test_mod.py::TestClass::test_method
```

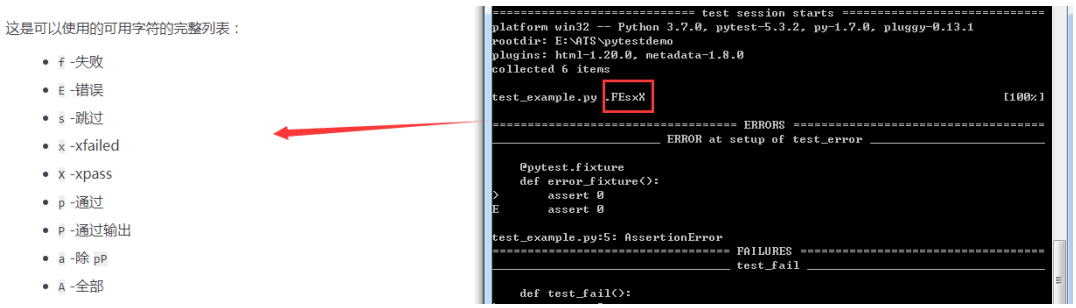
2、通过标记表达式运行测试

```
pytest -m slow
```

将运行用@Pytest.mark.slow 装饰器装饰的所有测试。

3、详细的总结报告

pytest – ra



4、分析测试执行持续时间

要获取最慢的 10 个测试持续时间的列表，请执行以下操作：

```
pytest --durations=10
```

5、将测试报告发送到在线 pastebin 服务

为每个测试失败创建一个 URL:

```
pytest --pastebin=failed
```

为整个测试会话日志创建一个 URL

```
pytest --pastebin=all
```

6、从 python 代码中调用 Pytest

```
pytest.main()
```

注意:

调用 Pytest.main()将导致导入你的测试及其导入的任何模块。由于 python 导入系统的缓存机制，Pytest.main()从同一进程进行后续调用不会反映两次调用之间对这些文件的更改。因此，Pytest.main()不建议从同一进程进行多次调用（例如，以重新运行测试）。

7、断言

使用标准 python assert 来验证 python 测试中的期望和值。

8、conftest.py

对于可能的值 scope 有：function，class，module，package 或 session。

9、Pytest 将建立一个字符串，它是用于在参数化 fixture，例如每个 fixtures 测试 ID。这些 ID 可以用于-k 选择要运行的特定情况，并且当一个故障发生时，它们还将标识特定情况。使用 Pytest 运行--collect-only 将显示生成的 ID。

10、使用直接测试参数化 fixture

给定测试文件的结构为:

```
tests/
    __init__.py
    conftest.py
        # content of tests/conftest.py
        import pytest
        @pytest.fixture
        def username():
            return 'username'
        @pytest.fixture
        def other_username(username):
```

```
            return 'other-' + username
    test_something.py
        # content of tests/test_something.py
        import pytest
        @pytest.mark.parametrize('username', ['directly-overridden-username'])
        def test_username(username):
            assert username == 'directly-overridden-username'
        @pytest.mark.parametrize('username', ['directly-overridden-username-other'])
        def test_username_other(other_username):
            assert other_username == 'other-directly-overridden-username-other'
```

11、使用非参数化的参数覆盖参数化 fixture

给定测试文件的结构为:

```
tests/
    __init__.py
    conftest.py
        # content of tests/conftest.py
        import pytest
        @pytest.fixture(params=['one', 'two', 'three'])
        def parametrized_username(request):
            return request.param
        @pytest.fixture
        def non_parametrized_username(request):
            return 'username'
    test_something.py
        # content of tests/test_something.py
        import pytest
        @pytest.fixture
        def parametrized_username():
            return 'overridden-username'
        @pytest.fixture(params=['one', 'two', 'three'])
        def non_parametrized_username(request):
            return request.param
        def test_username(parametrized_username):
            assert parametrized_username == 'overridden-username'
        def test_parametrized_username(non_parametrized_username):
            assert non_parametrized_username in ['one', 'two', 'three']
    test_something_else.py
        # content of tests/test_something_else.py
        def test_username(parametrized_username):
            assert parametrized_username in ['one', 'two', 'three']
        def test_username(non_parametrized_username):
```

```
assert non_parametrized_username == 'username'
```

12、monkeypatch

简单的 api 获取例子

```
# contents of app.py, a simple API retrieval example
import requests
def get_json(url):
    """Takes a URL, and returns the JSON."""
    r = requests.get(url)
    return r.json()
```

此外，如果该模拟程序旨在应用于所有测试，则 fixture 可以将其移动到 conftest.py 文件中并使用 with autouse=True 选项。

13、设置捕获方法或禁用捕获

有两种 pytest 执行捕获的方法:

- 文件描述符（FD）级别捕获（默认）: 将捕获对操作系统文件描述符 1 和 2 的所有写操作。
- sys 级别捕获: 仅写入 Python 文件，sys.stdout 并且 sys.stderr 将被捕获。不捕获对文件描述符的写入。

您可以从命令行影响输出捕获机制:

```
pytest -s                 # disable all capturing
pytest --capture=sys # replace sys.stdout/stderr with in-mem files
pytest --capture=fd    # also point filedescriptors 1 and 2 to temp file
```

14、内部 Pytest 警告

（1）类 PytestWarning

基类: UserWarning。

Pytest 发出的所有警告的基类。

（2）类 PytestAssertRewriteWarning

基类: PytestWarning。

Pytest 断言重写模块发出的警告。

（3）类 PytestCacheWarning

基地：PytestWarning。

缓存插件在各种情况下发出的警告。

（4）类 PytestCollectionWarning

基类：PytestWarning。

Pytest 无法在模块中收集文件或符号时发出警告。

（5）类 PytestConfigWarning

基地：PytestWarning。

针对配置问题发出警告。

（6）类 PytestDeprecationWarning

基类：Pytest.PytestWarning，DeprecationWarning。

在将来的版本中将删除的功能的警告类。

（7）类 PytestExperimentalApiWarning

基类：Pytest.PytestWarning，FutureWarning。

警告类别，用于表示 Pytest 中的实验。请谨慎使用，因为 API 可能会更改，甚至在将来的版本中会完全删除。

（8）类 PytestUnhandledCoroutineWarning

基类：PytestWarning。

当 Pytest 遇到作为协程的测试函数时发出警告，但任何异步感知插件均未处理该警告。本机不支持协程测试功能。

（9）类 PytestUnknownMarkWarning

基类：PytestWarning。

使用未知标记会发出警告。

1）doctest

就像普通的一样 conftest.py，fixtures 是在目录树 conftest 中发现的。这意味着，如果将 doctest 与源代码一起放入，则相关的 conftest.py 需要位于同一目录树中。fixtures

不会在同级目录树中发现!

2）跳过和 xfail

一个 xfail 意味着你期望测试失败的某些原因。一个常见的示例是对尚未实现的功能或尚未修复的错误进行测试。如果尽管测试通过但预期会失败（标记为 pytest.mark.xfail），则为 xpass，并将在测试摘要中报告。

Pytest 分别统计和列出跳过和 xfail 测试。默认情况下，不显示有关跳过/未通过测试的详细信息，以避免混乱输出。

```
pytest -rxXs    # show extra info on xfailed, xpassed, and skipped tests
```

17、如何在不同情况下跳过模块中的测试的快速指南:

1）无条件跳过模块中的所有测试:

```
pytestmark = pytest.mark.skip("all tests still WIP")
```

2）根据某些条件跳过模块中的所有测试:

```
pytestmark = pytest.mark.skipif(sys.platform == "win32", reason="tests for linux only")
```

3）如果缺少某些导入，请跳过模块中的所有测试:

```
pexpect = pytest.importorskip("pexpect")
```

18、跨测试运行

该插件提供了两个命令行选项，以从上次 Pytest 调用重新运行失败:

- --lf，--last-failed-仅重新运行失败。
- --ff，--failed-first-先运行故障，然后测试的其余部分

最后一次运行没有测试失败，或者找不到缓存的 lastfailed 数据，

Pytest 则可以使用以下--last-failed-no-failures 选项之一将该选项配置为运行所有测试或不运行测试:

```
pytest --last-failed --last-failed-no-failures all       # run all tests (default behavior)
pytest --last-failed --last-failed-no-failures none      # run no tests and exit
```

19、Pytest 支持 unittest 开箱即用地运行基于 Python 的测试

到目前为止，Pytest 不支持以下功能:

- load_tests 协议；

- 子测试

1）具有在给定上下文中自动使用的 fixture，@pytest.fixture(autouse=True)

2）xunit 样式：在每个模块/类/功能的基础上实现 fixture

Module setup/teardown

```
def setup_module(module):
 """ setup any state specific to the execution of the given module."""
def teardown_module(module):
""" teardown any state that was previously setup with a setup_module
    method.
     """
```

Class setup/teardown

```
@classmethod
def setup_class(cls):
""" setup any state specific to the execution of the given class (which
    usually contains tests).
    """
@classmethod
def teardown_class(cls):
""" teardown any state that was previously setup with a call to
    setup_class.
    """
```

方法和功能的 setup/teardown

```
def setup_method(self,method):
""" setup any state tied to the execution of the given method in a
    class.   setup_method is invoked for every test method of a class.
    """
def teardown_method(self,method):
      """ teardown any state that was previously setup with a setup_method
    call.
    """
def setup_function(function):
```

```
    """ setup any state tied to the execution of the given function.
    Invoked for every test function in the module.
    """

def teardown_function(function):
""" teardown any state that was previously setup with a setup_function call.
    """
```

3）工具启动时插件发现顺序

通常最好将 conftest.py 文件保存在顶级测试或项目根目录中。

20、典型 setup.py 摘录:

```
setup(..., entry_points={"pytest11": ["foo = pytest_foo.plugin"]}, ...)
```

21、通过名字来访问另一个插件

插件想要与另一个插件的代码协作，则可以通过插件管理器获取引用

```
plugin = config.pluginmanager.get_plugin("name_of_plugin")
```

22、Pytest API

23、caplog（）

访问和控制日志捕获。

捕获的日志可通过以下属性/方法获得:

```
* caplog.messages        -> list of format-interpolated log messages
* caplog.text            -> string containing formatted log output
* caplog.records         -> list of logging.LogRecord instances
* caplog.record_tuples   -> list of (logger_name, level, message) tuples
* caplog.clear()         -> clear captured records and formatted log output string
```

24、pip 用于安装应用程序和所有依赖项以及 Pytest 程序包本身。这样可确保您的代码和依赖项与系统 Python 安装隔离。

25、Pytest.approx

26、失败的视频/屏幕截图 Pytest-splinter、Pytest-bdd

27. 考虑以下文件和目录布局:

```
root/
|- foo/
```

```
|- __init__.py
|- conftest.py
|- bar/
    |- __init__.py
    |- tests/
        |- __init__.py
        |- test_foo.py
```

执行时会执行以下所有的目录文件

```
pytest root/
```

28、测试模块名称不能相同

29、从中找到 rootdir 的算法 args:

- 确定指定的公共祖先目录，这些目录 args 被识别为文件系统中存在的路径。如果找不到此类路径，则将公共祖先目录设置为当前工作目录。
- 寻找 Pytest.ini，tox.ini 并 setup.cfg 在父目录和文件向上。如果匹配，它将成为 ini 文件，并且其目录将成为 rootdir。
- 如果未找到 ini 文件，请 setup.py 从公共祖先目录向上查找以确定 rootdir。
- 如果没有 setup.py 被发现，寻找 Pytest.ini，tox.ini 并 setup.cfg 在每个指定 args 向上。如果匹配，它将成为 ini 文件，并且其目录将成为 rootdir。
- 如果找不到 ini 文件，则使用已经确定的公共祖先作为根目录。这允许在不属于包且没有任何特定 ini 文件配置的结构中使用 Pytest。

二、Pytest 实际运用

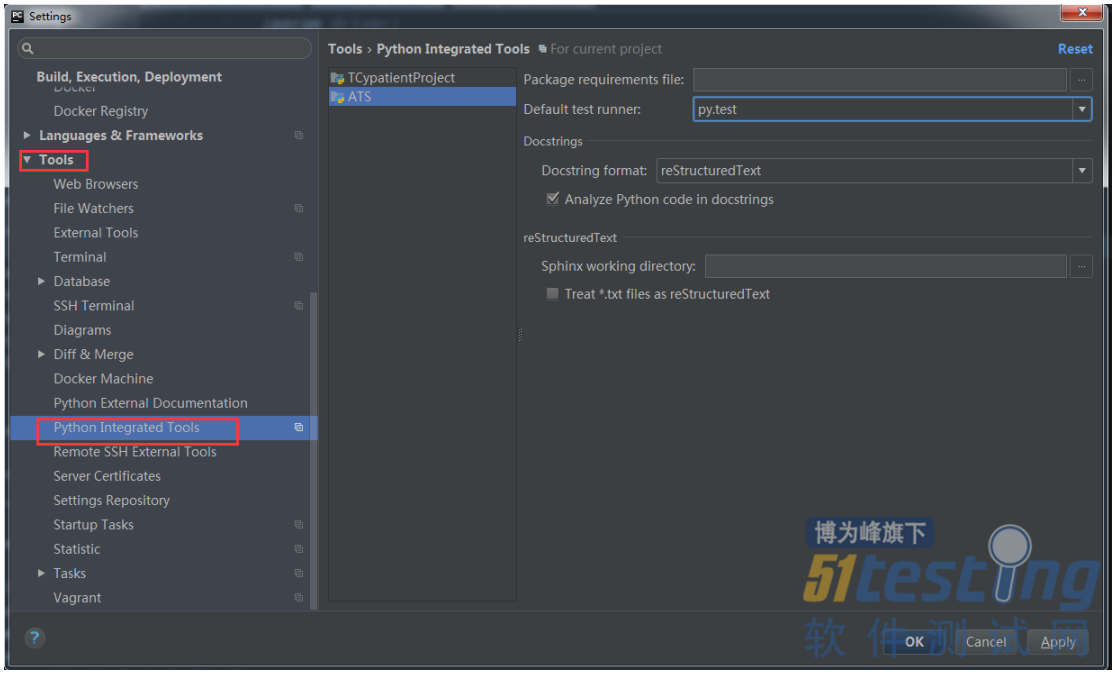
命令行中执行

```
pip install pytest
pip show pytest
```

```
C:\Users\admin>pip install pytest
Requirement already satisfied: pytest in e:\program files (x86)\python\python37\
lib\site-packages (4.4.0)
Requirement already satisfied: attrs>=17.4.0 in e:\program files (x86)\python\py
thon37\lib\site-packages (from pytest) (19.1.0)
Requirement already satisfied: setuptools in e:\program files (x86)\python\pytho
n37\lib\site-packages (from pytest) (39.0.1)
Requirement already satisfied: colorama; sys_platform == "win32" in e:\program f
iles (x86)\python\python37\lib\site-packages (from pytest) (0.4.1)
Requirement already satisfied: atomicwrites>=1.0 in e:\program files (x86)\pytho
n\python37\lib\site-packages (from pytest) (1.3.0)
Requirement already satisfied: py>=1.5.0 in e:\program files (x86)\python\python
37\lib\site-packages (from pytest) (1.7.0)
Requirement already satisfied: six>=1.10.0 in e:\program files (x86)\python\pyth
on37\lib\site-packages (from pytest) (1.12.0)
Requirement already satisfied: pluggy>=0.9 in e:\program files (x86)\python\pyth
on37\lib\site-packages (from pytest) (0.9.0)
Requirement already satisfied: more-itertools>=4.0.0; python_version > "2.7" in
e:\program files (x86)\python\python37\lib\site-packages (from pytest) (7.0.0)

C:\Users\admin>pip show pytest
Name: pytest
Version: 4.4.0
Summary: pytest: simple powerful testing with Python
Home-page: https://docs.pytest.org/en/latest/
Author: Holger Krekel, Bruno Oliveira, Ronny Pfannschmidt, Floris Bruynooghe, Br
ianna Laugher, Florian Bruhin and others
Author-email: None
License: MIT license
Location: e:\program files (x86)\python\python37\lib\site-packages
Requires: pluggy, more-itertools, attrs, py, six, atomicwrites, setuptools, colo
rama
Required-by: pytest-metadata, pytest-html

C:\Users\admin>
```

想要在 Pycharm 环境中测试用例以 Pytest 形式运行，可以这样设置。

Settings->Tools->Python Integreated Tools，选择默认的 test runner 为 “py.test”。



Pytest 生成 html 报告，命令行中安装 Pytest-html 插件。

pip install pytest-html

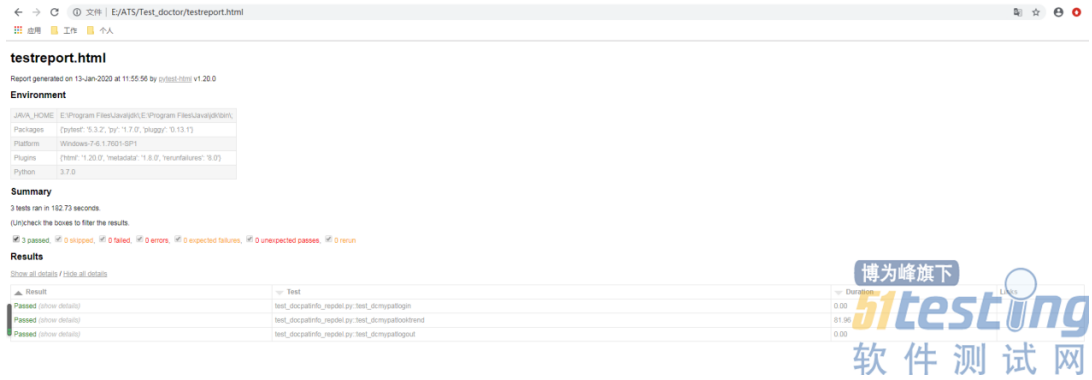
```
C:\Users\Administrator>pip3 install pytest-html
Requirement already satisfied: pytest-html in c:\python37\lib\site-packages (2.0
.1)
Requirement already satisfied: pytest>=5.0 in c:\python37\lib\site-packages (fro
m pytest-html) (5.3.5)
Requirement already satisfied: pytest-metadata in c:\python37\lib\site-packages
(from pytest-html) (1.8.0)
Requirement already satisfied: colorama; sys_platform == "win32" in c:\python37\
lib\site-packages (from pytest>=5.0->pytest-html) (0.4.1)
Requirement already satisfied: atomicwrites>=1.0; sys_platform == "win32" in c:\
python37\lib\site-packages (from pytest>=5.0->pytest-html) (1.3.0)
Requirement already satisfied: importlib-metadata>=0.12; python_version < "3.8"
in c:\python37\lib\site-packages (from pytest>=5.0->pytest-html) (1.5.0)
Requirement already satisfied: py>=1.5.0 in c:\python37\lib\site-packages (from
pytest>=5.0->pytest-html) (1.8.0)
Requirement already satisfied: more-itertools>=4.0.0 in c:\python37\lib\site-pac
kages (from pytest>=5.0->pytest-html) (7.0.0)
Requirement already satisfied: packaging in c:\python37\lib\site-packages (from
pytest>=5.0->pytest-html) (20.1)
Requirement already satisfied: wcwidth in c:\python37\lib\site-packages (from py
test>=5.0->pytest-html) (0.1.8)
Requirement already satisfied: pluggy<1.0,>=0.12 in c:\python37\lib\site-package
s (from pytest>=5.0->pytest-html) (0.13.1)
Requirement already satisfied: attrs>=17.4.0 in c:\python37\lib\site-packages (f
rom pytest>=5.0->pytest-html) (19.1.0)
Requirement already satisfied: zipp>=0.5 in c:\python37\lib\site-packages (from
importlib-metadata>=0.12; python_version < "3.8"->pytest>=5.0->pytest-html) (2.1
.0)
Requirement already satisfied: six in c:\python37\lib\site-packages (from packag
ing->pytest>=5.0->pytest-html) (1.12.0)
Requirement already satisfied: pyparsing>=2.0.2 in c:\python37\lib\site-packages
 (from packaging->pytest>=5.0->pytest-html) (2.4.6)

C:\Users\Administrator>
```

cmd 中执行 >pytest test_***.py --html=./testreport.html

```
C:\Users\admin>e:

E:\>cd ats\test_doctor

E:\ATS\Test_doctor>pytest test_docpatinfo_repdel.py --html=./testreport.html
============================= test session starts =============================
platform win32 -- Python 3.7.0, pytest-5.3.2, py-1.7.0, pluggy-0.13.1
rootdir: E:\ATS\Test_doctor
plugins: html-1.20.0, metadata-1.8.0, rerunfailures-8.0
collected 3 items

test_docpatinfo_repdel.py ...                                            [100%]

----------- generated html file: E:\ATS\Test_doctor\testreport.html -----------
======================== 3 passed in 182.77s (0:03:02) ========================

E:\ATS\Test_doctor>
微软拼音 - 新体验 2010 半 :
```

在报告存放路径下，可以用 Chrome 浏览器打开本地 html 链接，如 file:///E:/ATS/Test_doctor/testreport.html。

Pytest 测试报告形式如下所示



Pytest 失败重跑机制。在 UI 自动化测试中，由于受到网络不稳定、appium server 等影响，不是每次正确的测试用例都能运行通过，于是使用 Pytest 的失败重跑提高自动化测试的稳定性。安装插件 Pytest-rerunsfailures，在命令行执行

pip install pytest-rerunsfailures

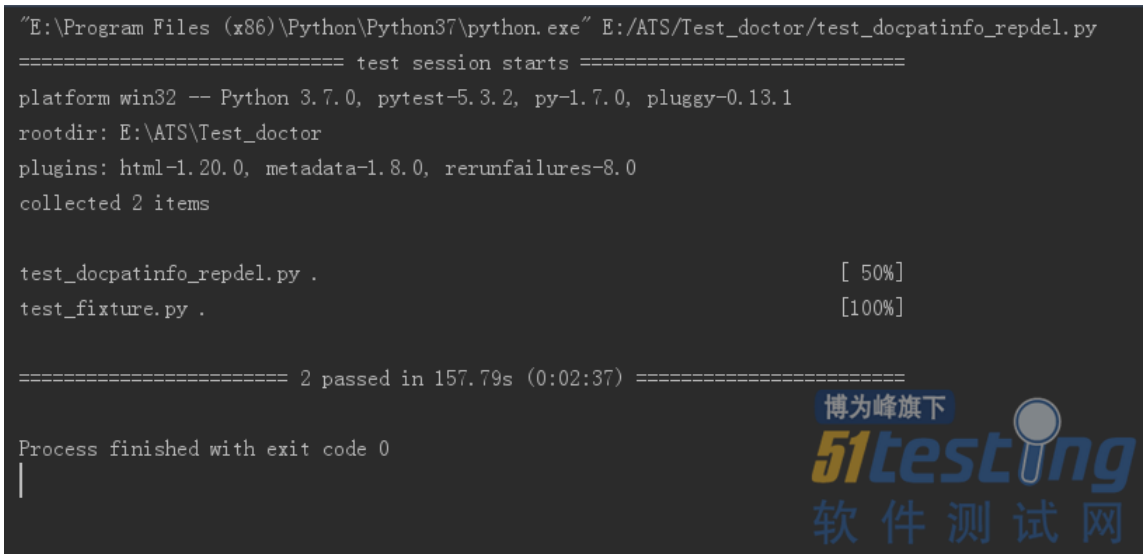
```
C:\Users\Administrator>pip3 install pytest-rerunfailures
Requirement already satisfied: pytest-rerunfailures in c:\python37\lib\site-pack
ages (8.0)
Requirement already satisfied: pytest>=4.4 in c:\python37\lib\site-packages (fro
m pytest-rerunfailures) (5.3.5)
Requirement already satisfied: setuptools>=40.0 in c:\python37\lib\site-packages
 (from pytest-rerunfailures) (45.2.0)
Requirement already satisfied: pluggy<1.0,>=0.12 in c:\python37\lib\site-package
s (from pytest>=4.4->pytest-rerunfailures) (0.13.1)
Requirement already satisfied: py>=1.5.0 in c:\python37\lib\site-packages (from
pytest>=4.4->pytest-rerunfailures) (1.8.0)
Requirement already satisfied: packaging in c:\python37\lib\site-packages (from
pytest>=4.4->pytest-rerunfailures) (20.1)
Requirement already satisfied: more-itertools>=4.0.0 in c:\python37\lib\site-pac
kages (from pytest>=4.4->pytest-rerunfailures) (7.0.0)
Requirement already satisfied: importlib-metadata>=0.12; python_version < "3.8"
in c:\python37\lib\site-packages (from pytest>=4.4->pytest-rerunfailures) (1.5.0
)
Requirement already satisfied: attrs>=17.4.0 in c:\python37\lib\site-packages (f
rom pytest>=4.4->pytest-rerunfailures) (19.1.0)
Requirement already satisfied: colorama; sys_platform == "win32" in c:\python37\
lib\site-packages (from pytest>=4.4->pytest-rerunfailures) (0.4.1)
Requirement already satisfied: wcwidth in c:\python37\lib\site-packages (from py
test>=4.4->pytest-rerunfailures) (0.1.8)
Requirement already satisfied: atomicwrites>=1.0; sys_platform == "win32" in c:\
python37\lib\site-packages (from pytest>=4.4->pytest-rerunfailures) (1.3.0)
Requirement already satisfied: six in c:\python37\lib\site-packages (from packag
ing->pytest>=4.4->pytest-rerunfailures) (1.12.0)
Requirement already satisfied: pyparsing>=2.0.2 in c:\python37\lib\site-packages
 (from packaging->pytest>=4.4->pytest-rerunfailures) (2.4.6)
Requirement already satisfied: zipp>=0.5 in c:\python37\lib\site-packages (from
importlib-metadata>=0.12; python_version < "3.8"->pytest>=4.4->pytest-rerunfailu
res) (2.1.0)

C:\Users\Administrator>
```

>pytest test_patlist.py --reruns 3 --html=./report.html

实现对测试用例重跑 3 次，3 次不通过才算失败，反之则运行成功。

实践得到，Pytest 不需要创建类，直接定义函数即可。

Pytest 和 unittest 最大的区别是不要求我们必须创建测试类，自己创建的测试类也不需要继承于 unittest.TestCase 类。但是 Pytest 要求我们创建的测试文件，测试类、方法、测试函数必须以“test”开头，Pytest 默认按照这个规则查找测试用例并执行它们。”



三、Pytest 避开的坑

1）PytestUnknownMarkWarning

解决方案:

① 若是单个标签

在 conftest.py 添加如下代码，直接拷贝过去，把标签名改成你自己的就行了

```
def pytest_configure(config):
config.addinivalue_line(
"markers", "login_success" # login_success 是标签名
)
```

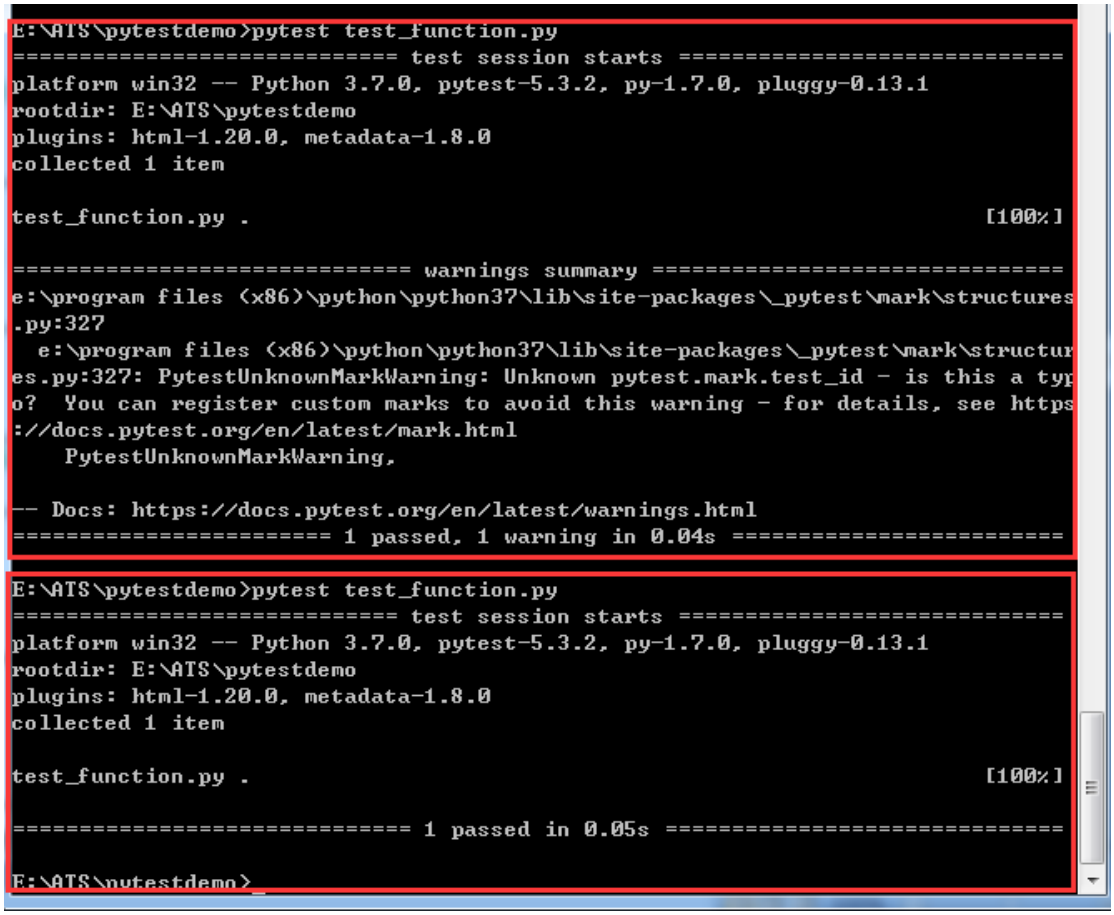
② 若是多个标签

在 conftest.py 添加如下代码，直接拷贝过去，把标签名改成你自己的就行了

```
def pytest_configure(config):
marker_list = ["testmark1","testmark2","testmark3"]  # 标签名集合
for markers in marker_list:
config.addinivalue_line(
"markers", markers
```

)

这样添加之后，再次运行，不再出现 warning。



2）UnicodeDecodeError: 'gbk' codec can't decode byte 0xae in position 42: illegal m

ultibyte sequence

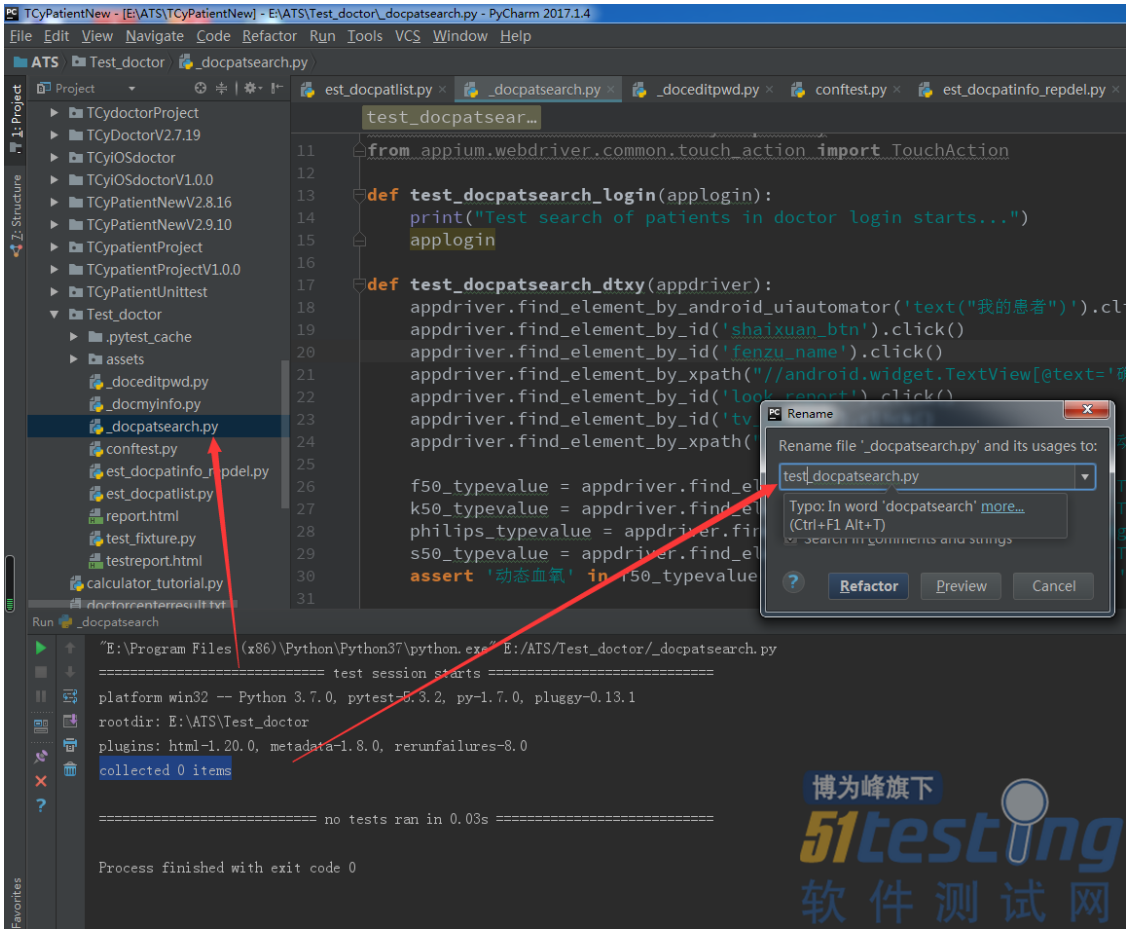
```
import pytest
@pytest.mark.parametrize("test_input,expected",[("3+5",8),("2+4",6),("6*9",42)])
def test_eval(test_input,expected):
    assert eval(test_input) == expected
```

解决办法：这个跟编码有关，代码目录中有个 Pytest.ini 文件，内容是这样:

```
[pytest]
doctest_encoding = UTF-8   #默认编码是 UTF-8
```

删除 Pytest.ini 文件之后，再次运行就可以了。

3）Pytest 执行用例时 collected 0 items



Pytest 执行的文件需要以 test 开头才能被查找到。

4）断言 assert

断言元素是否存在，例如

```
element = appdriver.find_element_by_id('icon_id')
assert element
```

Python 接口测试实践之用例封装及测试报告生成

◆ 作者：桃 子

今天以天气 API 接口为例，使用 python 语言实现用例编写、封装及报告生成功能

API 信息:

天气 API: http://www.51testing.com/?action-viewnews-itemid-4465288

URL:http://t.weather.sojson.com/api/weather/city/101030100

请求方式：get

参数：city 城市名称

一 、代码实现查询北京的天气信息

步骤:

1、新建 weather_api_test.py 文件

代码实现

```
#-*-coding:GBK -*-
import requests
from pip._vendor.requests.models import Response
url='http://t.weather.sojson.com/api/weather/city/101030100'
r=requests.get(url)
response_data=r.json()
print(r.text)
```

返回结果:

```
pydev debugger: starting (pid: 1068)
Finding files... done.
Importing test modules ... {"message":"success感谢又拍云(upyun.com)提供CDN赞助","status":
200,"date":"20200312","time":"2020-03-12 15:37:44","cityInfo":{"city":"天津市","cit
ykey":"101030100","parent":"天津","updateTime":"14:45"},"data":{"shidu":"31%","pm2
5":52.0,"pm10":84.0,"quality":"良","wendu":"10","ganmao":"极少数敏感人群应减少户外活动","foreca
st":[{"date":"12","high":"高温 13℃","low":"低温 3℃","ymd":"2020-03-12","week":"星期四",
"sunrise":"06:29","sunset":"18:15","aqi":69,"fx":"东北风","fl":"4-5级","type":"阴","n
```

二、用例集成到 Unittest

1、针对不同的参数场景进行测试

2、设置断言判断执行结果是否符合预期

实现原理:

首先导入 requests 库、unitest 、时间库

其次，创建天气 class 类

然后，分别创建 4 个函数，分别实现存放路径、正常传参、异常传参、缺省参数功能

3、用例设计

场景	描述	结果
正常参数	传入正常参数	提示'success 感谢又拍云(upyun.com)提供 CDN 赞助'
异常参数	传入异常参数，英文字符	提示'获取失败'
参数缺省	不传参数	提示'Request resource not found.'

代码实现:

新建 weather_api_unitest.py 文件

```
#-*-coding:GBK -*-
import unittest
import requests
from time import sleep
class weathertest(unittest.TestCase):
    def setUp(self):
        self.url='http://t.weather.sojson.com/api/weather/city/101030100'
        self.url_error='http://t.weather.sojson.com/api/weather/city/101030101'
        self.url_no='http://t.weather.sojson.com/api/weather/city'
```

```
    #参数正常
    def test_weather_tianjing(self):
        r=requests.get(self.url)
        result=r.json()
        self.assertEqual(result['status'],200)
        self.assertEqual(result['message'],'success 感谢又拍云(upyun.com)提供 CDN 赞助')
        sleep(3)
    #参数异常
    def test_weather_param_error(self):
        r=requests.get(self.url_error)
        result=r.json()
        self.assertEqual(result['status'],400)
        self.assertEqual(result['message'],'获取失败')
        sleep(3)
    #参数缺省
    def test_weather_no_param(self):
        r=requests.get(self.url_no)
        result=r.json()
        self.assertEqual(result['status'],404)
        self.assertEqual(result['message'],'Request resource not found.')
        sleep(3)
    if __name__=='_main_':
        unittest.main()
```

三、测试报告生成

1、创建文件夹如图，把测试用例放到 test_case 目录下

2、下载 BSTestRunner 模块并放置到 python Lib 文件夹下

如路径 C:\Python34\Lib

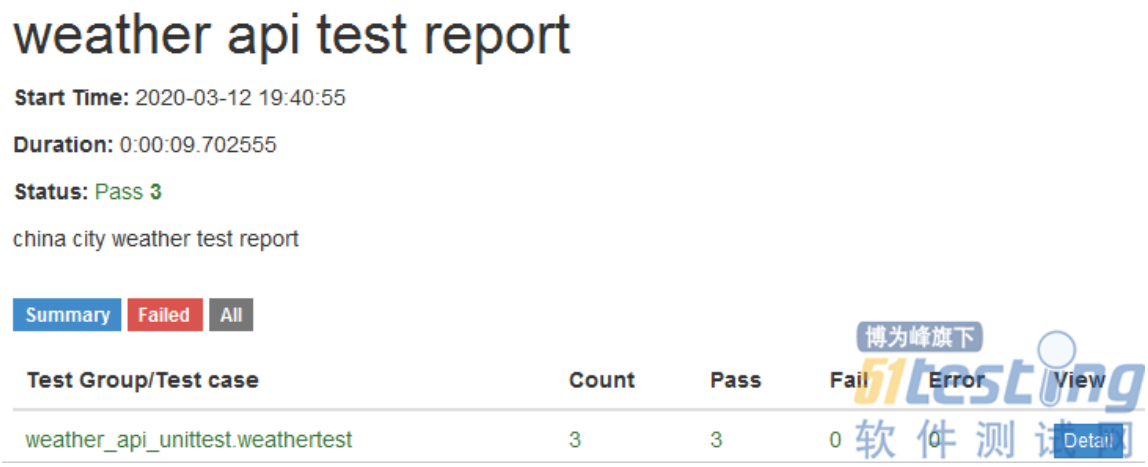
3、创建 run.py 文件

代码:

```
import unittest
from BSTestRunner import BSTestRunner
```

```
import time
#指定测试用例和报告路径
test_dir='./test_case'
report_dir='./reports'
#加载测试用例
discover=unittest.defaultTestLoader.discover(test_dir, pattern='weather_api_unittest.py')
#定义报告格式
now=time.strftime('%Y-%m-%d %H_%M_%S')
report_name=report_dir+'/'+now+'test_report.html'
#运行用例并生成测试报告
with open(report_name,'wb') as f:
    runner=BSTestRunner(stream=f,title="weather api test report",description="china city weather test report")
    runner.run(discover)
```

4、运行 run.py，在 reports 文件夹下查看生成的报告



■搜狗测试总监“微访谈”：测试员如何成长为团队核心>>https://dwz.cn/YyTF4Wks

人与机器人如何协同来测试软件

◆ 译者：枫　叶

摘要： 我们都从一些非常聪明的人那里听说过用于软件测试的 AI，但是这个想法引起了很多困惑。本文解决了你可能会问的一些问题,比如使用 AI 进行软件测试是否需要成为天才？AI 会取代我们成为测试员吗？在我们的测试策略中 AI 适合哪些方面？用训练狗的简单类比，了解 AI 如何适合测试。

我们都曾经听过或读过一些非常聪明的人关于用于软件测试的 AI 的知识，但是对此想法有很多困惑。

首先，我要说的是，我对数学和编码没有足够的热情，足以关心 AI 的内在和技术方面。我认为作为一个软件测试人员，它对技术概念有很好的认知，并且可以使用多种编程语言编写普通的代码。我相信软件测试是由人来完成的，并且我相信测试是通过探索和试验来了解产品来评估产品的过程。但是我也喜欢自动化和工具。

让我们解决你可能会问的一些问题，以便我们所有人都能更好地了解 AI 如何用于测试。

使用 AI 进行软件测试是否需要成为天才？

如上面所述，在 AI 的内部运作方面，我绝不是 AI 专家。信不信由你，由于我在狗训练方面的背景，我发现自己喜欢 AI。

严肃地说，让我们超级简化一下，把 AI 机器人与狗进行比较。如果你曾经养过小狗，我相信你有过这样的经验，当你要求让它们坐下来，但是它们只用困惑的脸抬头看着你。所以你掏出了点心，当它们的屁股撞到地上时，你就奖励了它们。它们很快了解到“坐下”等于报酬。然后，你开始在上面贴上标签。你会说“坐下”，如果屁股撞到了地面，它们会得到一种奖励，所以“坐下”意味着在地面上撞屁股，得到食物了！

同样，人工智能机器人以一种杂乱的方式抓取你的应用程序，尝试不同的路径并浏览不同的屏幕。只有当机器人开始为它们的行为获得“奖励”时，它们才开始学习我们

要它们做什么。一旦它们浏览了应用程序的一部分，它们便获得了奖励，我们标记了该动作，然后它们以可重复的方式执行了该动作。就是这么简单，无论你是用户还是培训师。

你不需要成为一名兽医就可以训练狗，但是对狗的思维方式和动机有一个大致的了解确实有帮助。同样，你无需能够创建 AI 机器人来了解如何使用它们进行软件测试。但是，你应该知道如何测试软件，并对 AI 的工作原理有足够的了解，以便正确使用它。

AI 会取代我们成为测试员吗？

继续这个比喻，我们直接说一个事实，那就是仍然有积极从事工作的训犬师。狗还没有学会，也似乎没有真正的欲望或意图去开始互相了解如何坐以待命。部分原因是它们不会说我们所能理解的语言。

机器人的相似之处在于它们尚无法自我训练。即使它们可以，它们也缺乏理解软件测试所需的上下文和目的的能力。因此，如果你认为你可以雇用很多 AI 机器人来代替你的测试人员，或者如果你认为自己将失去 AI 机器人的工作，那么你是完全错误的。

AI 在我们的测试策略中适合哪些方面？

如果你问自己这个问题，你将获得奖牌。关于人工智能，我一直注意到的准入障碍之一是二进制的想法：你要么必须选择“人工智能所有事物！”或“没有适合你的人工智能！”

任何事物都会贴近事实。正如我已经解释的那样，AI 不会取代软件测试员。我还想解决那个问题，就是 AI 无法而且不应该做所有事情。

AI 和自动化是可以在软件测试中使用的工具。它们不能自己测试软件，很容易被滥用，但是应使用它们来补充你的测试。就像私人助理一样，它们会执行我们没有时间或渴望去做的事情，这使我们有时间专注于重要的事情。

许多公司都试图“使所有事情自动化”。我们许多人仍在尝试使所有事情自动化。有些事情不应该自动化。安吉·琼斯（Angie Jones）对此进行了精彩的演讲和授课。就像自动化一样，人工智能不应该用于“AI 所有事物！”你可以将脚趾伸进众所周知的 AI 水里。

以狗训练类比为基础，你不会要求狗开汽车。我不会要求 AI 进行复杂的组合自动化。

你可能会想：“但是等一下。人工智能很聪明，所以我不应该让它去做困难的事情吗？” 答案是，你绝对可以让它完成艰巨的任务。但是你必须一遍又一遍地做些什么呢？你知道，这些东西令人无聊，高度可重复并且需要测试人员花费很多个小时吗？那真的是对测试人员大脑的最佳利用吗？你是否雇用它们只是日复一日地按按钮，还是雇用它们来做它们熟练的工作，这就是测试？

我供职的一家公司花费大量资金支付我的测试团队在生产环境中进行冗余 UI 检查。这是我们大多数人都不喜欢的令人麻木的工作。使用 AI，我们能够连续运行这些检查，并为开发和运营提供快速反馈。最重要的是，我们正在使用的 AI 服务提供的指标包括 CPU 使用率，内存使用率和性能。因此，在运行自动化程序的同时，我们能够对一些性能指标进行趋势分析并查看模式，而无需进行额外的特定负载和性能测试。这是一个处理我们问题的非常有效的解决方案。

人工智能的最佳用途可能只是处理测试人员不想做的所有事情。如果让机器人执行简单、可重复的任务，则可以让测试人员进行实际测试。因此，你不必担心每天都要按动按钮，而是要让它们动脑筋。

现在，你有了一个机器人，它在做测试人员以前在做的事情，除了更快、更有效之外，并且你有一个热情的人在热情地进行测试。恭喜，你刚刚增加了你的产品范围！

这就是为什么需要将 AI 纳入测试策略的原因，就像将自动化纳入测试策略一样。

总之，人工智能和人类可以而且应该一起测试软件。使用 AI 进行软件测试不必成为天才。你只需要对此保持灵活。人工智能不是软件测试的灵丹妙药，因此不应被这样使用。通过战略性地使用 AI 来补充其它测试工作，你可以大大提高产品覆盖率。而且，如果你使用 AI 来完成简单、可重复的任务，则可以让测试人员专注于它们擅长的工作：测试，从而提高测试人员的士气。

互联网江湖中的质量债

◆作者：多则惑少则明

摘要： 互联网江湖中，技术同学想必对技术债都不陌生，几乎每个研发团队，都会经历一次，甚至几次的技术重构，来减少技术债。同样，对于质量团队而言，或者对于测试同学而言，质量债也是不得不面对的一个挑战了，不仅仅是影响测试效率，甚至会直接导致测试场景的遗漏，进而引发不同程度的线上问题。下面就自己一些经验和感悟，聊聊自己的一些体会。

一、前言

深处互联网圈的同学，特别是技术方向的同学，想必对技术债一词耳熟能详了。“技术债”是 Ward Cunningham 在 1992 年提出的，它主要用来描述理想中的解决方案和当前解决方案中间的差距所隐含的潜在成本。类似金融债务：为了解决短期的资金压力，获得短期收益，个人或企业向银行或他人借款，从而产生债务，这种债务需要付出的额外代价是利息。

技术债的几个成因:

- xxx 必须按时上线;
- 当前先赶进度，稍后做一版重构;
- 团队内缺少必要的 code review;
- 做技术设计时可扩展性差

那么与技术债类似的，实际的项目中，往往还伴随有质量债。质量债，顾名思义，与质量相关的潜在问题/已存在问题。技术债，造成的是研发新需求时的痛苦连连，比如，不断跳坑、在原基础代码上做扩展困难、需要时不时重构等。而质量债，往往涉及质量传承问题，整体质量保障过程，是不断填质量债，又不断埋下新的质量债的过程。下面就自己的一点经验，来说说自己对项目中的质量债的一些体会，不足之处，欢迎大家交流指正。

二、质量债的几大表现

如果要判断一个质量团队是不是存在质量债，不妨回答一下下面的几个问题:

想了解某个模块的业务、测试方案、自动化方案等等情况，是否直接有文档/工具沉淀;

来了位新同学，需要做哪些事可以对项目规范/流程、整体测试方案熟悉起来;

模块 A，测试同学 A 刚刚测试完上线；之后的项目 B，测试同学 B 需要多久可以了解模块 A 及之前的测试方法;

总结起来，质量债的表现为:

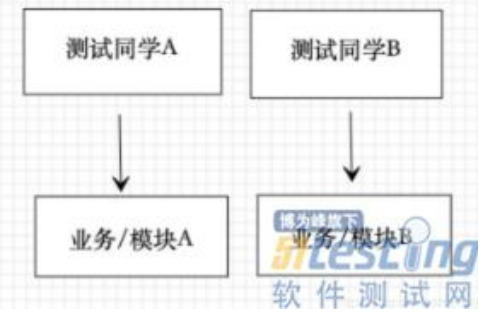
- 质量无良好沉淀
- 新同学介入困难
- 质量传承手段差

三、团队合作模式对质量债的影响

当前各个公司的质量团队中，团队模式大致可以分为几种:

1、纵向合作模式

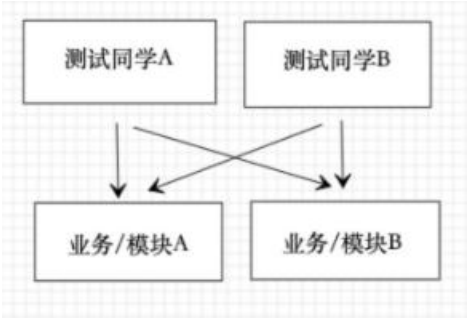
在这种模式下，测试同学之间耦合的工作比较少，除非特殊情况，突发状况，一般情况下，测试同学 A 只会负责业务/模块 A 的测试，测试同学 B 只会负责业务/模块 B 的测试，二人不会互换测试，如下图所示。



由于测试同学 A，测试同学 B“独测”一块内容，假设 A，B 同学都有不同程度的质量债，但 A 同学的质量债，不会让 B 同学背，反之亦然。当然在一些特殊的情况下，比如 A 同学去负责其他业务了，那么 A 同学留下的质量债就会由其接任者背负了。

2、交叉合作模式

有些团队，或许因为人手问题，或者 boss 想人员互为 backup，或者业务本身迭代模式决定，总之测试同学之间的工作会存在重合的部分。比如，测试同学 A，B 可能同时测试业务/模块 A，也可能上次测试同学 A 测试业务/模块 A，测试同学 B 测试业务/模块 B，这次互换模块进行测试了，如下图所示。



在这种模式下，也同样假设 A,B 同学都有不同程度的质量债。那么 A 同学的质量债，会被 B 同学背负，B 同学的质量债，也会给 A 同学背负。于是，该团队同学只能对业务一知半解了：A 同学跟进过业务 A,B 的测试，之后 B 同学又跟进过业务 A,B 的测试，A,B 同学循环往复跟进业务，但却由于互相对各自的项目不熟悉，导致 A,B 同学都不能说 100%熟悉业务 A,B。

因而，团队质量债与整体团队质量合作的模式密切相关。 相比纵向合作模式， 交叉合作模式下的团队，会受质量债的影响更大。如果想让团队同学合作更加顺畅，交叉合作模式下的团队，更加需要采取手段来尽可能减少，或提前规避质量债。

四、质量债的防治

让质量债可衡量，是其防治的关键。团队在具体操作的时候，可根据其导致的几个关键问题，来有针对性的量化。

比如，针对每个项目，沉淀各自的业务、技术、测试方案等等；针对业务，提取核心业务场景及其介绍；针对测试手段，形成整体性的工具等等。

1、质量债的预防

要想让团队尽可能少的欠质量债，除了靠自觉以外，可能需要团队里面形成质量传承的氛围。比如，每个项目后的分享机制，详细文档沉淀机制等等。类似技术债一样，需要从上到下而推进。

比如，A 同学负责上线一个业务/模块 A 后，给团队其他同学把项目相关沉淀输出出来。那么下次，B 同学想回归业务/模块 A 的时候，不会完全不懂其修改，或不必靠口口相传的方式去问 A 同学，B 同学的工作效率，甚至是效果会提升不少。

2、质量债的治理

关于治理，可能针对不同的团队特点，方法多多。但众多方法，有一个根本的目标：让质量保障方案，在团队内流转起来，尽可能保证大家对业务、测试方案的理解一致。

具体治理方案，大致有两个方向：

方案 1：技术性方案

为了更好避免理解性偏差，效率提升，采用尽可能多的自动化工具，来避免质量债问题。比如，A 同学跟进项目，测试上线了模块 A， 那么 A 同学需要提供类似一键自动化测试的工具，之后 B 同学再回归测试模块 A 的时候，直接用自动化工具测试。

方案 2：分享性方案

如果某块业务，不适合自动化测试，或者项目排期紧迫等等，可以采用组内分享机制，来尽可能避免质量债问题。比如，同样的，A 同学跟进项目，测试上线了模块 A，那么 A 同学及时组织一个分享会，把业务、技术、测试方案等等内容，同步给组内其他同学。

五、写在最后

交叉合作模式下的质量团队，或多或少都会存在不同程度的质量债，在质量债重的团队中，测试同学做项目往往是十分痛苦的，需要找若干同学去沟通，xxx 业务目前在线上是什么样子的，之前怎么造场景的等等；更危险的是，一旦存在当前测试同学感知之外的场景，改动，往往就会引发不同程度的线上问题。

质量债，同技术债一样，是伴随业务的不断迭代，而日积月累起来的。预防需要持续性的时间和精力投入，治理也需要从上到下的共同合作。

或许，质量债不可能 100%消除，但为了整体的质量，需要尽可能减少质量债。

做测试的你，职业倦怠了么？

◆ 作者：咖啡猫

“大半的人在二十岁或三十岁上就死了。一过这个年龄，他们只变了自己的影子。以后的生命不过是用来模仿自己，把以前真正有人味儿的时代所说的，所做的，所想的，所喜欢的，一天天的重复，而且重复的方式越来越机械，越来越脱腔走板。” ——《约翰·克里斯多夫》罗曼·罗兰

一、测试同行们的烦恼

“嘀嘀...”

上午十点，测试交流群里热闹了起来。

“各位老铁，有事请教。我毕业后在一家新能源的外企做软件测试快两年了，刚开始的时候一切还是挺新鲜的，每天上班都是干劲满满的。从业务需求的理解、测试方案的定制、测试环境的搭建到测试用例的执行、BUG 闭环管理、测试报告的提交，我一样样地跟着师傅学感觉也是很有成就感的。但是工作满满上手后，现在反倒越来越提不起精神了，公司的产品线单一，技术更新也很慢，基本业务熟悉后，会一点点 linux 指令能把系统搭建起来，再按测试方案在系统上点点点就够了。当初的学习热情也没有了，想到以后都是每个月拿着 5K 的薪水日复一日地点点点，日子没啥盼头，就内心很失落也很烦躁。我是不是该换个环境了，你们谁那有合适的坑啊，求带求推荐...”新人群友小李一通抱怨着.....

“求带+1。你还好，至少刚毕业没有房贷，一人吃饱全家不饿。我们公司从去年开始业绩缩水的厉害，公司领导本想 2020 年开辟一下新业务看看能不能扭转局面，结果又赶上了新冠肺炎疫情，这不前两天视频会议上老板直接宣布全员降薪 50%，如果能撑过去的话，差额年底再补发，否则只能走破产清算了.....最近上班一点精神提不起来，想到每个月这点薪水，月月要还房贷，一家老小都指望着你，真的是压力山大，况且还

要 996 地加班，年纪大了，身体也吃不消了.....”资深群友老陈吐槽道。

“你们谁有我惨，说出去是个国企体制内吧，但是人际关系复杂的很。领导啥都不会还喜欢瞎指挥，本来没啥事非得让我们测试留下来陪开发、产品加班。和其他组合作做个项目，出了问题就甩锅给我们测试组，真到论功行赏的时候，结果隔壁组的领导抢功劳，用他个人名义报了奖，奖金我们一分钱没有，可把我恶心坏了。我现在就想去个人际关系简单点的地方呆着。在这上班比上坟心情还沉重哇，时间一久非抑郁不可.....”群友 CC 说道。

一时间，原本清净的微信群开始热闹了起来，我们很多做测试的小伙伴们，都会有如同上面三位群友一样的感觉吧。比如:

上班提不起精神，上班就想下班
莫名焦虑担心，看到领导发的消息就心慌
啥活都不想干，啥人也不想理
感觉身体被掏空，情感已经枯竭
并不是不想活，只是不想上班而已
.....

我想这些小伙伴们大概是“职业倦怠”了

二、什么是职业倦怠

美国纽约大学心理学家弗登伯格（H. J. Freudenberger）研究发现，许多社会服务性职位需要较多的情绪性工作，面对较多人际压力源，长年精力耗损，使得工作热诚容易消退，进而产生对人漠不关心以及对工作持负面态度。比如护士、教师、银行柜员等

1974 年，其在《职业心理学》杂志上首次提出了职业倦怠（bum-out）这个词语。

他认为，职业倦怠是指个体在体力、精力和能力上都无法应付外界的要求而产生的身心疲劳与耗竭的状态。

前述例子中，无论是小李的缺乏学习动力和工作成就感、老陈的身体吃不消、薪水不满意还是 CC 所说的人际关系复杂难以应付，都可以归结为他们在体力、精力、能力上无法应付外界需求而产生的职业倦怠。

如果你也职业倦怠了，莫慌！因为在当今社会，大部分人都逃不过职业倦怠。美国心理学会（American Psychological Association）的数据是:53%的人认为工作让他们极度疲累。《中国“工作倦怠指数”调查报告》显示有 7 成的中国职场人轻微倦怠，13％的人重度职业倦怠。

三、职业倦怠的影响

职业倦怠对身体和心理有双重影响，并且会形成心理健康影响身体健康，身体健康再反过来影响心理健康的恶性循环。轻则只是心情不太好、情绪不稳定、浑浑噩噩度日，重则失眠、头痛、抑郁、免疫力低下等等。

四、职业倦怠的成因

职业倦怠的成因很复杂，我认为分为外部因素和内部因素两方面。

就外部因素而言，美国社会心理学家马斯拉琪（Maslach）和加拿大组织心理学家米

歇尔·P·莱特（Michael P. Leiter）共同提出了“职业倦怠的工作匹配理论”，认为“职业倦怠”感的产生，与六个方面的“不匹配”有着密切关联:

1、工作负荷大：比如一些长期 996 的人群，特别是 35 岁之后，熬夜通宵出差加班的情况就越来越感到吃不消了。

2、控制：控制中的不匹配与职业倦怠中的无力感有关，通常表明个体对工作中所需的资源没有足够的控制，或者指个体对使用他们认为最有效的工作方式上没有足够的权威。比如处处受制于人，无法按照自己的认为可行的方案去解决问题，而是迫于无奈按领导或者客户的意愿事倍功半地干活，工作上没有主动权也没有成就感。

3、报酬：这里包括物质奖励和精神奖励两方面，比如小李因为没有成就感（精神奖励）而职业倦怠，而老陈因为没有充足的物质奖励而职业倦怠。

4、社交：比如复杂的人际关系，同事之间相互甩锅，领导外行领导内行，客户刁钻难搞等等.....CC 就是这种情况，她觉得上班比上坟的心情还要沉重得多。

5、公平：这里特指由于工作量分配或者报酬分配上引起的不公平。比如每天混日子的关系户升官发财，每天加班干活的老黄牛一直得不到重用，那么老黄牛们就很容易倦怠。

6、价值观冲突：这种不多见，但是也有。比如开发组和测试组，开发组觉得目前只要主流程是好的，能让客户凑合用就可以了，毕竟市场不等人；测试则认为重量重于泰山，BUG 真要到客户那里去了，损失的还是公司的名誉.....

“一切是考验，看你如何着眼；万事在说法，试你如何用心。”外部因素是导火索，内部因素才是根本。某些心理学研究也表明，具有一些性格特征的人更容易产生职业倦怠。比如:

1、外控者：指那些遇到困难时习惯于把责任推向外部，而不是寻求问题的解决途径的人。他们相信社会的安排，命运和运气等因素决定了自己的状况，而自己的努力无济于事，倾向于放弃对自己生活后果负责。比如：CC 觉得人际关系复杂，领导同事难相处都是别人的错，和自己为人处世无关。

2、低自尊者：低自尊者低总是需要通过外界的肯定和鼓励来确定自己的价值，他们希望自己做的每件事都有及时的反馈，如果一旦外界没有给到他们足够多认可和鼓励，他们很在逆境中前行和自我驱动，他们也更容易出现职业倦怠。比如小李每天点点

点，得不到领导同事的认可，自己也慢慢产生无价值感。

3、抑郁型人格：这类人比较敏感，觉得人生空虚毫无意义可研，他们更爱关注事务的消极影响。

五、职业倦怠的解决之道

1、加强体育锻炼。工作之余，多做有益身心的体育活动。可以是慢跑、瑜伽、舞蹈、搏击操等等。毕竟身体才是革命的本钱，干 IT 的经常为了赶项目进度而熬夜加班出差开会等，如果没有好身体根本就抗不下来。并且我们在工作中积攒的一些负能量也可以通过体育锻炼的方式宣泄出来，体育锻炼可以帮助我们大脑分泌多巴胺，让我们的情绪得到舒缓。比如老陈，如果他平时多加锻炼，估计也不会因为身体吃不消而产生职业倦怠。

2、培养兴趣爱好。人生不止是工作，俗话说得好“不要把鸡蛋放在一个篮子里”。有的人对工作过于认同，一旦工作上稍有不如意，就认为自己是个毫无价值的人，进而产生职业倦怠。培养兴趣爱好，就是为了丰富业余生活，降低对工作的过度的认同感。

3、设置合理地目标，不断提升自己。经常性对自己进行职业规划，设置合理地目标，不断提升自己的职业技能，一旦目标达成就适当鼓励自己。这样就能防止浑浑噩噩度日的局面发生。比如文中的小李，她目前只是掌握了功能测试的基础内容，她还可以自学性能测试、自动化测试、接口测试的东西，提高自己的技能，等待更好的职业机会。

4、重新认识自己，学会乐观应对一切。人生不如意十有八九，每个人的职场生涯都或多或少会有抱怨和遗憾。但是你不能被这些不如意蒙蔽了双眼，而是应该客观认识自己的优点、缺点、能力、兴趣以及这份工作的优点、缺点。而不是一味只盯着工作的缺点看。一个人的幸福感其实和自我定位也很有关系，如果梦想和现实有差距，要么奋起直追要么降低期望，抱怨是一点用也没有的。

5、谋定而后动，换一份更合适的工作。当你自我定位清晰并且认清了现任工作的本质后，如果确实人岗的匹配度不高，你可以去搜寻更合适的工作，为跳槽转行提前做准备，而不是盲目地裸辞。

■搜狗测试总监“微访谈”：测试员如何成长为团队核心>>https://dwz.cn/YyTF4Wks

数据处理类系统功能测试探索

◆作者：雷陈芳

随着大数据、数据挖掘、人工智能等技术的快速发展，企业愈加重视数据资产的价值，积极打造从数据采集、汇总、处理加工至应用的数据应用闭环。数据质量是数据应用领域建设的重中之重，数据处理类系统是数据质量的重要影响因素，面对繁杂海量的数据与加工规则，系统测试如何在时间与资源的限制下保证质量与效率值得探索。本文介绍了一套数据处理类系统的功能测试方法，并提出测试执行策略，以提高测试质量与测试效率。

一、引言

信息技术的广泛应用使得企业生产经营变得更加高效，同时也沉淀了大量的数据。随着大数据、数据挖掘、人工智能等技术的快速发展，企业愈加重视数据资产的价值，将数据中隐藏的信息与规律转化为企业的生产力，使数据成为企业的决策依据。当前，数据应用能力已成为企业的核心竞争力，支撑经营决策、客户营销、产品创新、绩效管理、风险管控、监管报送及信息披露等诸多场景。因此，各企业积极打造从数据采集、汇总、处理加工至应用的闭环，建设大数据平台、数据仓库、数据集市、数据湖等各种形式的“数据中台”，实现海量数据的快速聚集汇总与处理加工，为企业发展增速提供动力。

本文所述数据处理类系统是指根据业务目标与需求对数据进行汇集、拼接、整合、统计、加工等直接处理的纯后台系统，例如大数据平台、数据仓库、数据集市、数据湖等。此类项目无前台界面，多为批量程序，由调度系统控制定期运行，处理结果以及数据文件或数据库表等形式存储，供下游应用使用。

数据是数据应用的基础，数据质量对于数据应用至关重要，数据处理类系统是数据质量的重要影响因素，测试作为系统质量保障的最后一道防线，面对繁杂海量的数据加工规则需求，如何在有限资源的条件下兼顾测试质量与测试效率值得探讨。

本文根据过往项目经验整理了一套数据处理类系统的功能测试方法，对其中重要的目标表检查进行详细梳理，根据检查点来源将目标表检查内容分为了技术层面与业务层面，建议测试人员按照从技术层面到业务层面、从简单到复杂、从宏观到微观的测试执行策略。

二、数据处理类系统功能测试

目前数据处理类系统常用的功能测试方法主要有两类，一是白盒测试，主要使用代码检查方法，由测试人员根据业务需求对系统批量程序的代码或脚本进行检查，较容易发现一些直观的问题，比如判断条件中的比较符号写反、判断条件的遗漏、边界值的遗漏等。此外，代码检查有助于加深测试人员对数据处理功能的理解，进行黑盒测试案例设计时更有针对性。

二是黑盒测试，即运行批量程序，在运行过程中检查是否出现报错信息与中断，运行结束后对生成的数据表或数据文件，即目标表检查。目标表检查是数据处理类系统测试最主要的内容，通过检查间接验证系统实现的加工逻辑是否正确满足业务需求。目标表检查一般是通过编写 SQL 语句查询的方式实现。本文根据检查点的来源将此部分内容分为技术层面和业务层面，详细阐述如下。

（一）技术层面检查内容

技术层面检查内容主要是对“接口”格式的检查。数据处理类系统的处理结果供下游应用使用，可以视作与下游系统的“接口”，因此最基本的要求是满足系统间“接口”格式约定以及因约定导致的技术约束。具体检查内容包括:

（1）目标表齐全，表名与约定一致;

（2）目标表字段齐全，字段名、字段类型、长度精度等属性与约定一致;

（3）目标表主键设置与约定一致，以及由此导致的技术约束即不存在主键重复的记录、不存在主键字段为空值的记录;

（4）时间拉链与约定一致，以及由此导致的技术约束即时间拉链不存在断链、倒链和交叉链。这部分测试内容执行简单，容易发现错误，且错误的影响范围广。

（二）业务层面检查内容

业务层面检查内容是根据业务规则与业务经验设计的检查，可以分为两部分，一是对根据业务规则与经验得到的简单业务约束的检查。具体包括:

（1）目标表记录数的数量级是否与业务经验一致，例如客户信息表的记录数大致与企业客户约十万人一致;

（2）目标表字段的空值比例满足业务预期，例如客户编号、交易金额等字段空值比例为 0，客户地址字段空值比例小于 50%;

（3）目标表字段的默认值比例满足预期，例如客户学历字段默认值为初中，业务预期客户学历为初中的记录数占比小于 30%;

（4）目标表字段数据取值范围是否合理，数值型字段的最小值、最大值在业务预期范围内，如取款金额满足预期取值范围（0，50000]，离散型字段的取值符合业务规定，简单的如客户性别、客户学历、年龄段等，复杂的如客户标签、产品标签等，文本型字段的字符串格式符合业务规定，如贵宾客户编号以 VIP 开头、时间字段满足 YYYY-MM-DD-HH;

（5）字段间简单的业务逻辑关系是否满足，例如开始时间早于结束时间、持有产品总数等于各类产品持有数之和。对简单业务约束的测试结果与测试数据密不可分，建议使用脱敏的历史数据，但需注意脱敏时不破坏数据的有效性。

二是对具体加工规则的检查，即测试人员根据具体加工规则，对输入数据处理类系统的源数据进行计算，获得正确计算结果，与目标表进行比对。对于记录数较少的目标表可以进行全表核对，对于记录数较多的目标表可以进行表级汇总属性核对，例如表记录数是否一致、数值型字段的表级加总值是否一致、离散型字段的分布比例是否一致等，进一步抽取部分记录进行核对，抽样时可根据等价类划分法、边界值法等有针对性的抽取。例如功能需求为汇总统计各子机构的 A 产品营销额，可先测试汇总后目标表记录数是否于子机构数量一致、目标表 A 产品营销额总计字段的表级加总值是否与 A 产品总营销额一致，测试通过后进一步从目标表中抽取小部分样本测试对应子机构的 A 产品营销额是否计算正确。

三、测试执行策略

在执行测试时可以按照从技术层面到业务层面、从简单到复杂、从宏观到微观的测试执行策略。技术层面的测试内容执行较简单耗时短，容易发现影响范围广的错误，例如批量程序执行后发现缺失某张目标表，即可说明批量程序存在问题，反馈开发人员进行排查并修复，无需进行业务层面的测试。从简单到复杂是指先进行简单业务约束的检查，进而执行具体复杂业务规则的检查。从宏观到微观，在具体业务规则检查中先执行表记录数、表级加总值等表级属性的核对，再抽样核对具体记录。

四、总结

本文介绍了一套数据处理类系统的功能测试方法，通过项目经验，将目标表检查内容分为技术层面与业务层面，其中技术层面主要进行接口格式以及主键约束、外键约束等技术约束的检查，业务层面主要进行取值范围、逻辑关系、加工规则等业务约束的检查。测试人员可遵循从技术层面到业务层面、从简单到复杂、从宏观到微观的测试执行策略，避免遗漏测试点，同时耗费最小代价尽可能多的发现缺陷，从而极大提高测试质量与测试效率。

Django 数据管理及接口测试

◆ 作者：桃 子

本文主要介绍了 Django 连接 mysql 操作，及测试用例封装，jekin 集成并通过邮件发送测试报告的全过程。

一、 Django 数据管理

通过数据管理可以解决接口之间数据间的相互干扰，导致断言失败时不知道是接口引起的还是数据引起的错误

数据场景:

测试数据库，将数据每次测试前初始化

1、安装 mysql 数据库

下载地址: https://dev.mysql.com/downloads/installer/

2、下载安装 navicat 数据管理工具

下载地址: https://www.navicat.com.cn/

3、Django 迁移 mysql

setting 配置文件修改如下信息:

```
DATABASES = {
    'default': {
        # 'ENGINE': 'Django.db.backends.sqlite3',
        # 'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
        'ENGINE':'Django.db.backends.mysql',
        'HOST':'127.0.0.1',
        'PORT':'3306',
```

```
        'NAME':'Django_restful',
        'USER':'root',
        'PASSWORD':'',
        'OPTIONS':{
            'isolation_level':None,
            'init_command':"SET sql_mode='STRICT_TRANS_TABLES'",
                }
    }
}
```

4、安装数据库驱动

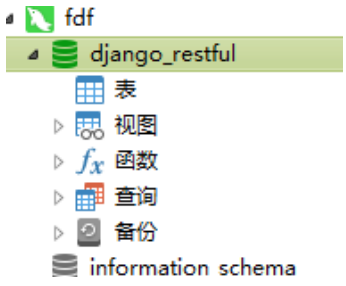
Django_restful 下的_init.py-文件修改代码如下:

```
import pymysql
pymysql.install_as_MySQLdb()
```

5、连接数据库

打开 navicat，点击连接新建如下图

新建名称 Django_restful 的数据库



6、创建 models

models 用来创建和存储数据

打开 api 下的 models.py，输入以下代码创建 user 和 group 表

```
from Django.db import models
# Create your models here.
class User(models.Model):
    username=models.CharField(max_length=100)
    email=models.CharField(max_length=100)
    groups=models.CharField(max_length=100)
    def _str_(self):
        return self.username
class Group(models.Model):
    name=models.CharField(max_length=100)
    def _str_(self):
        return self.name
```

7、导入 models

在 serializers.py 和 views.py 文件导入 models，同时去掉 Django 默认的数据库

serializers.py

```
# from django.contrib.auth.models import User,Group
from rest_framework import serializers
from api.models import User,Group
```

views.py

```
1  from django.shortcuts import render
2  #from django.contrib.auth.models import User,Group
3  from rest_framework import viewsets
4  from api.serializers import UserSerializer,GroupSerializer
5  from api.models import User,Group
6
```

8、数据库迁移

在 cmd 输入：python manage.py makemigrations api

```
D:\django_restful>python manage.py migrate
C:\Python34\lib\site-packages\pymysql\cursors.py:170: Warning: (3
DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO' sql mod
ed with strict mode. They will be merged with strict mode in a fu

  result = self._query(query)
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
```

python manage.py migrate

如果提示 access denied error，很有可能 setting 文件中 password 不一致

```
pymysql.err.OperationalError: (1045, "Access denied for user 'root'@'localhost'
(using password: NO)")
```

python manage.py createsuperuser

```
D:\django_restful>python manage.py createsuperuser
C:\Python34\lib\site-packages\pymysql\cursors.py:170: War
DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO'
ed with strict mode. They will be merged with strict mode

  result = self._query(query)
Username (leave blank to use 'administrator'): root
Email address: 635907412@qq.com
Password:
Password (again):
This password is too short. It must contain at least 8 ch
Password:
Password (again):
Superuser created successfully.
```

重启服务网页登录，创建用户返回 navicat 查看数据，如果存在说明迁移成功

python manage.py runserver

没有问题请忽略这一步

问题 1：误把数据库里的表格删除，重新迁移会失败

解决方案：

把整个数据库删除 Django_restful，然后重新迁移执行上面的命令

9、封装初始化操作

包括：数据连接、清除、插入、关闭数据库

在 api 下面新建目录 test_project，然后新建 sql_action.py，输入下面的命令

```
from pymysql import connect
import yaml
import logging
class DB():
    def __init__(self):
        logging.info('==================init data===============')
        logging.info('connect db...')
        self.conn=connect(host='127.0.0.1',user='root',password='xinsheng2',db='Django_restful')
    def clear(self,table_name):
        logging.info('clear db...')
        clear_sql='truncate '+table_name+';'
        with self.conn.cursor() as cursor:
            cursor.execute('set foreign_key_checks=0;')
            cursor.execute(clear_sql)
        self.conn.commit()
    def insert(self,table_name,table_data):
        logging.info('inser data...')
        for key in table_data:
            table_data[key]="'"+str(table_data[key])+"'"
        key=','.join(table_data.keys())
        value=','.join(table_data.values())
        logging.info(key)
        logging.info(value)
        insert_sql='insert into '+table_name+'('+key+')'+'values'+'('+value+')'
        logging.info(insert_sql)
        with self.conn.cursor() as cursor:
            cursor.execute(insert_sql)
        self.conn.commit()
    def close(self):
        logging.info('close db')
```

```
            self.conn.close()
            logging.info('==============init finished!============')
        def init_data(self,datas):
            for table,data in datas.items():
                self.clear(table)
                for d in data:
                    self.insert(table,d)
            self.close()
if __name__ == '__main__':
    db=DB()
    # db.clear('api_user')
    # db.clear('api_group')
    # user_data={'id':1,'username':'zxw2018','email':'zxw2018@163.com'}
    # db.insert('api_user',user_data)
    # db.close()
    f=open('datas.yaml','r')
    datas=yaml.load(f)
    db.init_data(datas)
```

10、封装初始化数据

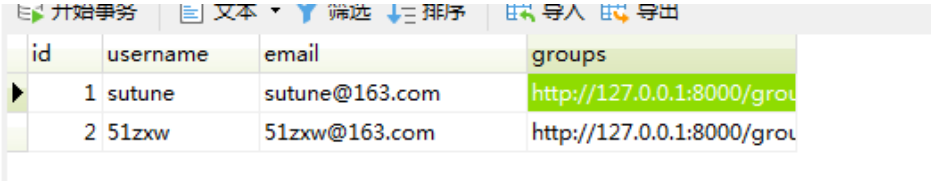
使用 yaml 来封装数据

如果没有安装 yaml 可以在 cmd 中输入 pip install pyyaml 安装

在 test_project 新建 datas.yaml 文件，输入下面语句

```
api_user:
  - id: 1
    username: sutune
    email: sutune@163.com
    api_user.groups: http://127.0.0.1:8000/groups/1/
  - id: 2
    username: 51zxw
    email: 51zxw@163.com
    api_user.groups: http://127.0.0.1:8000/groups/2/
api_group:
  - id: 1
    name: Developer
  - id: 2
    name: Tester
```

运行 sql_action.py，查看数据库内容如下图



id	username	email	groups
1	sutune	sutune@163.com	http://127.0.0.1:8000/grou
2	51zxw	51zxw@163.com	http://127.0.0.1:8000/grou

问题 1：遇到一个问题无论怎么运行都提示“Ran 0 tests in 0.000s”，查了好多资料刚开始以为是代码写错了，后来发现原来是运行错误

右键选择 run as ->python run 就 ok 了

问题 2:

执行 mysql_action.py 时报错：pymysql.err.ProgrammingError: （1064, "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'groups） values （'1','sutune','sutune@163.com','http://127.0.0.1:8000/groups/1/'）' at line 1"）。是 datas.yaml 文件里的 groups 与 MySQL 的关键词冲突，导致执行 insert 语句报错，如何解决

解决方案：把 datas_yaml 文件里面的 groups 字段改成：api_user.groups 亲测有效

11、测试用例封装

在 test_project 新建 test_Django_restful.py 文件，实现增删改查操作，每次回归就不用担心数据环境问题了，在文件中输入下面语句

```
import requests
import unittest
from mysql_action import DB
import yaml
from Django.contrib.auth.context_processors import auth

class UserTest(unittest.TestCase):
    def setUp(self):
        self.base_url='http://127.0.0.1:8000/users'
        self.auth=('51zxw','zxw20182018')

    def test_001_get_user(self):
        r=requests.get(self.base_url+'/1',auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'sutune')
        self.assertEqual(result['email'], 'sutune@163.com')
```

```
    def test_002_add_user(self):
        form_data={'id':3,'username':'zxw666','email':'51zxdddw@163.com','groups':'http://127.0.0.1:80
00/groups/2'}
        r=requests.post(self.base_url+'/',data=form_data,auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'zxw666')

    def test_003_delete_user(self):
        r=requests.delete(self.base_url+'/2/',auth=self.auth)
        self.assertEqual(r.status_code,204)

    def test_004_update_user(self):
        form_data={'email':'wx@162.com'}
        r=requests.patch(self.base_url+'/1/',auth=self.auth,data=form_data)
        result=r.json()
        self.assertEqual(result['email'],'wx@162.com')
#
#     def test_005_no_auth(self):
#         r=requests.get(self.base_url)
#         result=r.json()
#         self.assertEqual(result['detail'],'Authentications creadentianls were not provided')
#
#
class GroupTest(unittest.TestCase):
    def setUp(self):
        self.base_url='http://127.0.0.1:8000/groups'
        self.auth=('51zxw','zxw20182018')

    def test_001_group_developer(self):
        r=requests.get(self.base_url+'/1/',auth=self.auth)
        result=r.json()
        self.assertEqual(result['name'],'Developer')

    def test_002_add_group(self):
        form_data={'name','Pm'}
        r=requests.post(self.base_url+'/',auth=self.auth,data=form_data)
        result=r.json()
        self.assertEqual(result['name'],'Pm')

    def test_003_updata_group(self):
        form_data={'name','boss'}
        r=requests.patch(self.base_url+'/2/',auth=self.auth,data=form_data)
```

```
        result=r.json()
        self.assertEqual(result['name'],'boss')
    def test_003_delete_group(self):
        r=requests.delete(self.base_url+'/1/',auth=self.auth)
        self.assertEqual(r.status_code,204)

    if __name__=='main()':
        db=DB()
        f=open('datas.yaml','r')
        datas=yaml.load(f)
        db.init_data(datas)
        unnttest.main()
```

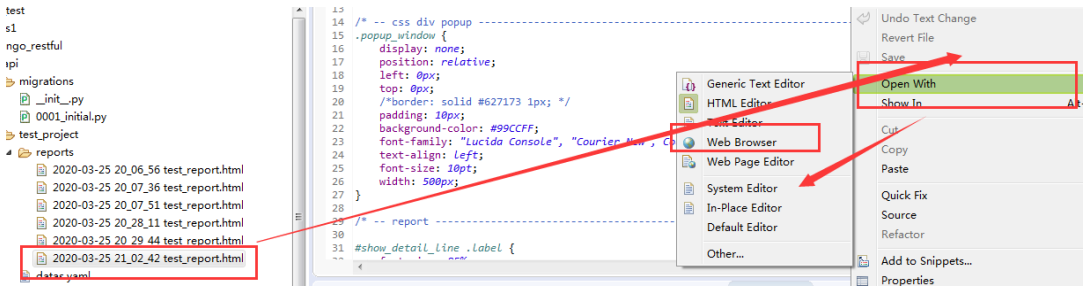
12、执行测试用例及测试报告生成

在 test_project 新建 report 文件夹（用于存放测试结果），新建 run.py 文件，写入一下代码

```
import unittest
from BSTestRunner import BSTestRunner
import time,yaml
from mysql_action import DB
db=DB()
f=open('datas.yaml','r',encoding='GBK')
datas=yaml.load(f)
db.init_data(datas)
test_dir='.'
report_dir='./reports'
discover=unittest.defaultTestLoader.discover(test_dir,pattern='test_Django_restful.py')
now=time.strftime('%Y-%m-%d %H_%M_%S')
report_name=report_dir+'/'+now+' test_report.html'
with open (report_name,'wb') as f:
    runner=BSTestRunner(stream=f,title='API Test Report',description='Django Restful API Test Report')
    runner.run(discover)
```

运行完成后，在 report 目录下查看生成的测试报告

注意：report 文件夹需要执行刷新操作才显示文件



Test Group/Test case	Count	Pass	Fail	Error	View
test_django_restful.GroupTest	4	2	0	2	Detail
test_002_add_group			error		
test_003_updata_group			error		
test_django_restful.UserTest	4	4	0	0	Detail
Total	8	6	0	2	

13、日志配置

通过配置日志信息可以快速帮我们定位问题

在 test_project 新建 log 文件夹（用于存放日志信息），新建 log.conf 文件，写入以下代码

```
[loggers]
keys=root,infoLogger
[logger_root]
level=DEBUG
handlers=consoleHandler,fileHandler

[logger_infoLogger]
handlers=consoleHandler,fileHandler
qualname=infoLogger
propagate=0

[handlers]
keys=consoleHandler,fileHandler

[handler_consoleHandler]
class=StreamHandler
level=INFO
formatter=form02
args=(sys.stdout,)

[handler_fileHandler]
```

```
class=FileHandler
level=INFO
formatter=form01
args=('./logs/runlog.log', 'a')
[formatters]
keys=form01,form02
[formatter_form01]
format=%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s
[formatter_form02]
format=%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s
```

将运行程序分别打上日志信息

run.py 引入配置文件

```
import unittest
from BSTestRunner import BSTestRunner
import time,yaml
from mysql_action import DB
import logging.config

CON_LOG='log.conf'
logging.config.fileConfig(CON_LOG)
logging=logging.getLogger()

db=DB()
f=open('datas.yaml','r',encoding='GBK')
datas=yaml.load(f)
db.init_data(datas)

test_dir='.'
report_dir='./reports'

discover=unittest.defaultTestLoader.discover(test_dir,pattern='test_Django_restful.py')

now=time.strftime('%Y-%m-%d %H_%M_%S')
report_name=report_dir+'/'+now+' test_report.html'

with open (report_name,'wb') as f:
    runner=BSTestRunner(stream=f,title='API Test Report',description='Django Restful API Test
Report')
    logging.info('====api test ======')
    runner.run(discover)
```

在 mysql_action.py 添加日志

```
from pymysql import connect
import yaml
import logging
class DB():
    def __init__(self):
        logging.info('==================init data===============')
        logging.info('connect db...')
        self.conn=connect(host='127.0.0.1',user='root',password='root',db='Django_restful')

    def clear(self,table_name):
        logging.info('clear db...')
        clear_sql='truncate '+table_name+';'
        with self.conn.cursor() as cursor:
            cursor.execute('set foreign_key_checks=0;')
            cursor.execute(clear_sql)
        self.conn.commit()

    def insert(self,table_name,table_data):
        logging.info('inser data...')
        for key in table_data:
            table_data[key]="'"+str(table_data[key])+"'"

        key=','.join(table_data.keys())
        value=','.join(table_data.values())

        logging.info(key)
        logging.info(value)

        insert_sql='insert into '+table_name+'('+key+')'+'values'+'('+value+')'
        logging.info(insert_sql)

        with self.conn.cursor() as cursor:
            cursor.execute(insert_sql)
        self.conn.commit()

    def close(self):
        logging.info('close db')
        self.conn.close()
        logging.info('==============init finished!============')

    def init_data(self,datas):
```

```
            for table,data in datas.items():
                self.clear(table)
                for d in data:
                    self.insert(table,d)
            self.close()

if __name__ == '__main__':
    db=DB()
    f=open('datas.yaml','r')
    datas=yaml.load(f)
    db.init_data(datas)
```

在 test_Django_restful.py 添加日志

```
import requests
import unittest
from mysql_action import DB
import yaml
from Django.contrib.auth.context_processors import auth
import logging

class UserTest(unittest.TestCase):
    def setUp(self):
        self.base_url='http://127.0.0.1:8000/users'
        self.auth=('51zxw','zxw20182018')

    def test_001_get_user(self):
        logging.info('test_0001_get_user')
        r=requests.get(self.base_url+'/1',auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'sutune')
        self.assertEqual(result['email'], 'sutune@163.com')

    def test_002_add_user(self):
        logging.info('test_0002_add_user')
        form_data={'id':3,'username':'zxw666','email':'51zxdddw@163.com','groups':'http://127.0.0.1:80
00/groups/2'}
        r=requests.post(self.base_url+'/',data=form_data,auth=self.auth)
        result=r.json()
        self.assertEqual(result['username'],'zxw666')

    def test_003_delete_user(self):
        logging.info('test_0003_delete_user')
        r=requests.delete(self.base_url+'/2/',auth=self.auth)
```

```
            self.assertEqual(r.status_code,204)

    def test_004_update_user(self):
        logging.info('test_0004_update_user')
        form_data={'email':'wx@162.com'}
        r=requests.patch(self.base_url+'/1/',auth=self.auth,data=form_data)
        result=r.json()
        self.assertEqual(result['email'],'wx@162.com')
#
#     def test_005_no_auth(self):
#         r=requests.get(self.base_url)
#         result=r.json()
#         self.assertEqual(result['detail'],'Authentications creadentianls were not provided')
#
#
class GroupTest(unittest.TestCase):
    def setUp(self):
        self.base_url='http://127.0.0.1:8000/groups'
        self.auth=('51zxw','zxw20182018')

    def test_001_group_developer(self):
        logging.info('test_0001_get_user')
        r=requests.get(self.base_url+'/1/',auth=self.auth)
        result=r.json()
        self.assertEqual(result['name'],'Developer')

    def test_002_add_group(self):
        logging.info('test_0002_add_user')
        form_data={'name','Pm'}
        r=requests.post(self.base_url+'/',auth=self.auth,data=form_data)
        result=r.json()
        self.assertEqual(result['name'],'Pm')

    def test_003_updata_group(self):
        logging.info('test_0003_delete_user')
        form_data={'name','boss'}
        r=requests.patch(self.base_url+'/2/',auth=self.auth,data=form_data)
        result=r.json()
        self.assertEqual(result['name'],'boss')

    def test_004_delete_group(self):
        logging.info('test_0004_delete_user')
```

```
        r=requests.delete(self.base_url+'/1/',auth=self.auth)
        self.assertEqual(r.status_code,204)

if __name__=='main()':
    db=DB()
    f=open('datas.yaml','r')
    datas=yaml.load(f)
    db.init_data(datas)
    unnttest.main()
```

运行 run.py 文件，在 log 文件夹下查看 log 信息

```
2020-03-25 21:45:44,354 mysql_action.py[line:8] INFO ==================init data========
2020-03-25 21:45:44,356 mysql_action.py[line:9] INFO connect db...
2020-03-25 21:45:44,454 mysql_action.py[line:13] INFO clear db...
2020-03-25 21:45:45,868 mysql_action.py[line:21] INFO inser data...
2020-03-25 21:45:45,870 mysql_action.py[line:28] INFO email,id,username,api_user.groups
2020-03-25 21:45:45,871 mysql_action.py[line:29] INFO 'sutune@163.com','1','sutune','htt
2020-03-25 21:45:45,873 mysql_action.py[line:32] INFO insert into api_user(email,id,user
2020-03-25 21:45:46,032 mysql_action.py[line:21] INFO inser data...
2020-03-25 21:45:46,034 mysql_action.py[line:28] INFO email,id,username,api_user.groups
2020-03-25 21:45:46,035 mysql_action.py[line:29] INFO '51zxw@163.com','2','51zxw','http:
2020-03-25 21:45:46,037 mysql_action.py[line:32] INFO insert into api_user(email,id,user
2020-03-25 21:45:46,180 mysql_action.py[line:13] INFO clear db...
2020-03-25 21:45:48,119 mysql_action.py[line:21] INFO inser data...
2020-03-25 21:45:48,121 mysql_action.py[line:28] INFO id,name
2020-03-25 21:45:48,124 mysql_action.py[line:29] INFO '1','Developer'
2020-03-25 21:45:48,127 mysql_action.py[line:32] INFO insert into api_group(id,name)valu
2020-03-25 21:45:48,294 mysql_action.py[line:21] INFO inser data...
2020-03-25 21:45:48,297 mysql_action.py[line:28] INFO id,name
2020-03-25 21:45:48,299 mysql_action.py[line:29] INFO '2','Tester'
2020-03-25 21:45:48,301 mysql_action.py[line:32] INFO insert into api_group(id,name)valu
2020-03-25 21:45:48,489 mysql_action.py[line:39] INFO close db
2020-03-25 21:45:48,493 mysql_action.py[line:41] INFO ==============init finished!=======
2020-03-25 21:45:49,674 run.py[line:29] INFO ====api test ======
2020-03-25 21:45:49,678 test_django_restful.py[line:58] INFO test_0001_get_user
2020-03-25 21:45:50,129 test_django_restful.py[line:64] INFO test_0002_add_user
2020-03-25 21:45:50,191 test_django_restful.py[line:72] INFO test_0003_delete_user
2020-03-25 21:45:50,238 test_django_restful.py[line:80] INFO test_0004_delete_user
2020-03-25 21:45:51,456 test_django_restful.py[line:15] INFO test_0001_get_user
2020-03-25 21:45:52,217 test_django_restful.py[line:22] INFO test_0002_add_user
2020-03-25 21:45:52,948 test_django_restful.py[line:30] INFO test_0003_delete_user
```

14、Jenkins 集成

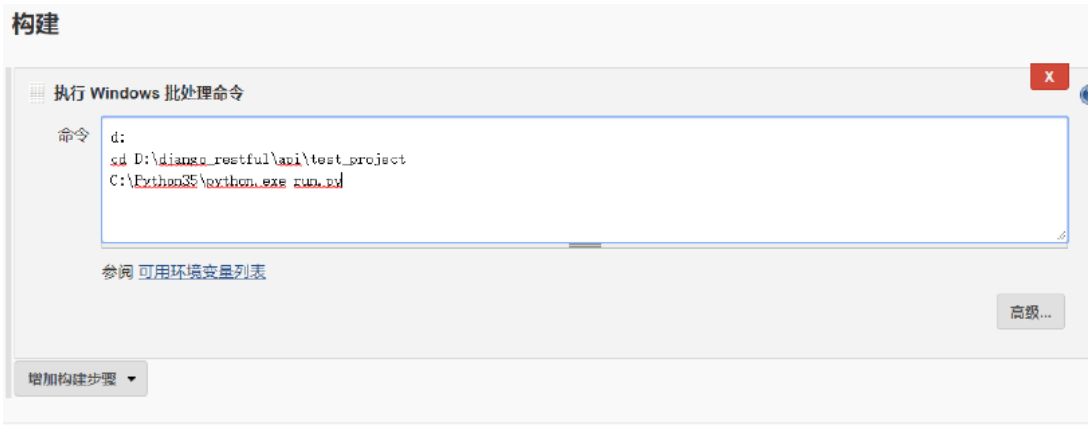
Jenkins 是一个自动化测试平台，持续集成（CI）的工具

安装包路径：链接：http://www.51testing.com/html/89/n-4465289.html

创建项目 Django_restful_api

构建中执行 windows 批处理程序

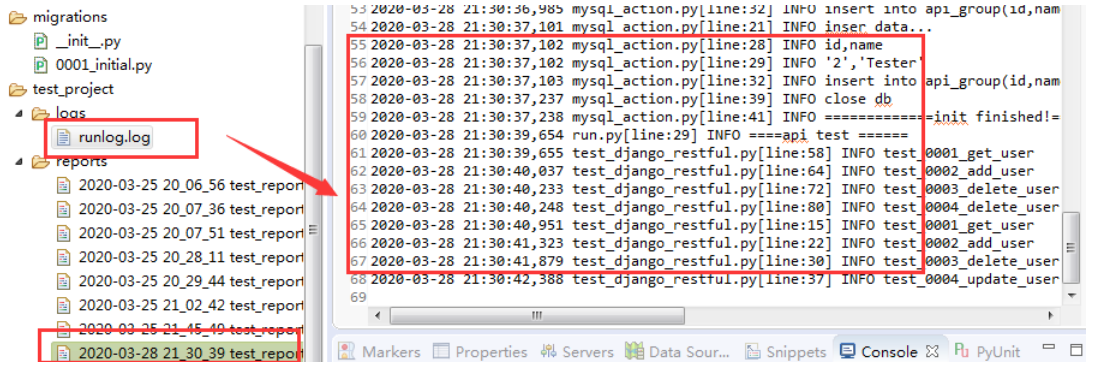
```
d:
cd D:\django_restful\api\test_project
C:\Python35\python.exe run.py
```



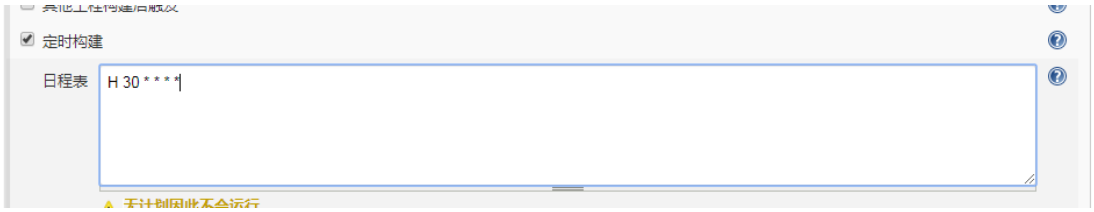
点击执行

控制台输出显示 success

返回程序中可以看到日志及报告内容



定时构建每 30 分钟执行一次



使用案例

每天下午 18 点定时构建一次

```
0 18 * * 1-5
```

每天早上 8 点构建一次

```
0 8 * * *
```

每 30 分钟构建一次:

```
H/30 * * * *
```


邮件发送

在 runl.py 文件，更新代码，实现发送带附件的邮件执行用例、更新数据库等功能

```
# -*- coding:utf-8 -*-
import unittest
from BSTestRunner import BSTestRunner
import time,yaml
from mysql_action import DB
import logging.config
import smtplib
from email.mime.text import MIMEText
from email.header import Header
import os

CON_LOG='log.conf'
logging.config.fileConfig(CON_LOG)
logging=logging.getLogger()

db=DB()
f=open('datas.yaml','r',encoding='GBK')
datas=yaml.load(f)
db.init_data(datas)

def send_mail(latest_report):
    f=open(latest_report,'rb')
    mail_content=f.read()
    f.close()
```

```
    smtpserver='smtp.163.com'
    # 发送邮箱用户名密码
    user='zxy13941778515@163.com'
    password='LSCCTNZSAQDNKWHO'

    sender='zxy13941778515@163.com'
    receives=['luoditao@126.com','zxy13941778515@163.com']
    # 发送邮件主题和内容
    subject = 'hello'
    # HTML 邮件正文
    msg = MIMEText(mail_content, 'html', 'utf-8')
    msg['Subject'] = Header(subject, 'utf-8')
    msg['From'] = sender
    msg['To'] = ','.join(receives)
    smtp = smtplib.SMTP_SSL(smtpserver, 465)
    # HELO 向服务器标识用户身份
    smtp.helo(smtpserver)
    # 服务器返回结果确认
    smtp.ehlo(smtpserver)
    # 登录邮箱服务器用户名和密码
    smtp.login(user, password)

    print("Start send Email...")
    smtp.sendmail(sender, receives, msg.as_string())
    smtp.quit()
    print("Send Email end!")

def latest_report(report_dir):
    lists = os.listdir(report_dir)
    # 按时间顺序对该目录文件夹下面的文件进行排序
    lists.sort(key=lambda fn: os.path.getatime(report_dir + '\\' + fn))
    print(("new report is :" + lists[-1]))

    file = os.path.join(report_dir, lists[-1])
    print(file)
    return file

if __name__ == '__main__':
    test_dir='.'
    report_dir='./reports'

    discover=unittest.defaultTestLoader.discover(test_dir,pattern='test_Django_restful.py')
```

```
now=time.strftime('%Y-%m-%d %H_%M_%S')
report_name=report_dir+'/'+now+' report.html'

with open (report_name,'wb') as f:

    runner=BSTestRunner(stream=f,title='API   Report',description='Django Restful API   Report')
    logging.info('====api test ======')
    runner.run(discover)
f.close()

#h 获取最新测试报告
latest_report=latest_report(report_dir)
#发送邮件报告
send_mail(latest_report)
```

打开邮箱查看测试报告:

API Report

Start Time: 2020-03-30 12:47:46

Duration: 0:00:02.947168

Status: Pass **6** Error **2**

Django Restful API Report

Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
test_django_restful.GroupTest	4	2	0	2	Detail
test_002_add_group				error	
test_003_updata_group				error	
test_django_restful.UserTest	4	4	0	0	Detail
Total	**8**	**6**	**0**	**2**	

错误: 运行 run 文件报错 554

smtplib.SMTPDataError: (554, b'DT:SPM 163 smtp3,G9xpCgC3vS0Qd4Fe_IpUAQ--.78S3 1585542929,please see http://mail.163.com/help/help_spam_16.htm?ip=60.16.239.253&hostid=smtp3&time=1585542929')

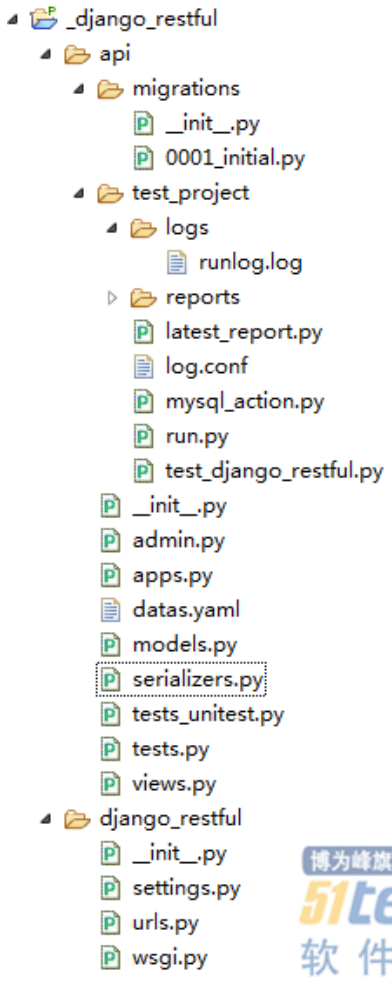
查看了网上的解决办法:

1）主题和附件内容不能包含 test

2）将收件人包含有发件人的邮箱

总结各文件实现的功能

回顾一下每个为念实现的功能是什么



Django-restful 文件夹

- setting.py 文件 实现 Django 数据迁移到 mysql
- _init_.py 实现安装数据库驱动功能
- urls.py 代码配置路由信息
- test_project 文件夹
- models.py 在数据库中创建 user 和 group 表的 model
- serializers.py 定义 API 返回形式，返回哪些字段，返回怎样的格式等

- views.py 文件　定义视图的展现形式，如何向用户展示数据，展示什么数据等
- datas.yaml 文件　使用 yaml 来封装数据
- test_Django_restful.py 文件 实现增删改查操作，用例封装功能
- run.py 文件 现发送带附件的邮件执行用例、生成测试报告、更新数据库等功能
- log.conf 文件　存放日志信息

《51 测试天地》(五十七) 下篇 精彩预览

马上阅读