目 录

关于网关接口参数有效性检查 case 设计自动化的思考

作者：咖喱

一、背景介绍

1.1 意义和价值

随着电子商务部门的扩张、部门业务的增加、对测试的质量要求有增无减，测试人员紧缺成为工作中的瓶颈问题。此时，将测试过程中繁琐、重复的测试流程自动化，减轻手工测试的劳动量，提高测试的效率具有很重要的研究意义。

自动化测试的好处归纳如下：

- 对新版本执行回归测试
- 更多更频繁的沉闷、耗时的测试
- 替代手工测试的困难
- 更好地利用时间
- 帮助缓解测试人员紧缺的问题
- 基于对三个网关（支付宝网关、工行网关、直充网关等）20 个左右的接口的观察统计发现：
- 对参数的有效性（如：参数 args　int 型，[2 6]）的测试用例达到 50%~70%。
- 此类测试用例的设计，具有很强的规律性，具体规律在 1.2 中作详细描述。
- 此类测试用例的结果也比较好把握，错误码一般就两种：参数缺失或者参数错误。
- 这些接口参数的独立性强，参数不会互相作用。

因此，基于以上特征，大多数接口的参数有效性检查的测试用例所需要的数据，可以用程序自动化生成，可以缩短测试中数据准备需要的部分时间，具有很好的应用价值。

1.2 可行性分析

首先，本文以网关的 charge_notify 接口为例子，charge_notify 接口的参数以及属性如下：

notify_id：	int、[1 ， 999999999] 、[必填、需要合法性检查]
notify_des:	string、长度 1~30 位、可能有火星文、[可选、需要合法性检查]
date：	date、格式 YYYYMMDDHHIISS [必填、需要合法性检查]
ip：	string [必填、但不对其进行合法性检查]
sign_type：	fix 固定的几个值[md5、des]
amount:	float、[0、无穷] [必、需要检查]

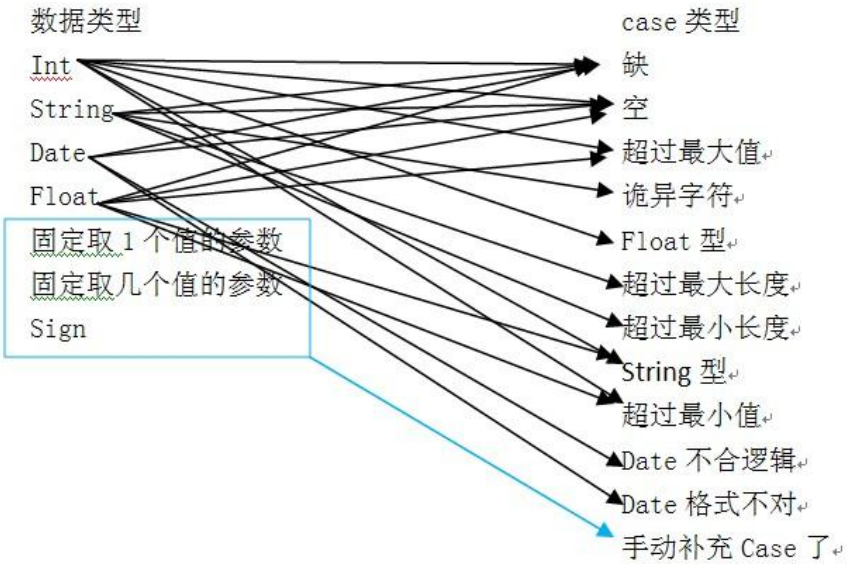
对应的测试用例如下：

缺、空、大于最大值、小于最小值。因为是[必填、需要合法性检查]所以这些用例对应错误码都是“参数缺失”或者“参数非法”。

缺、空、火星文、长度小于最小值、长度大于最大值。因为是[可选、需要合法性检查]所以前三个用例对应结果是正确的，后面两个对应错误码是“参数非法”。

.....

测试用例就不一一列出，罗列下参数类型和用例总类，对应关系如下图 1：

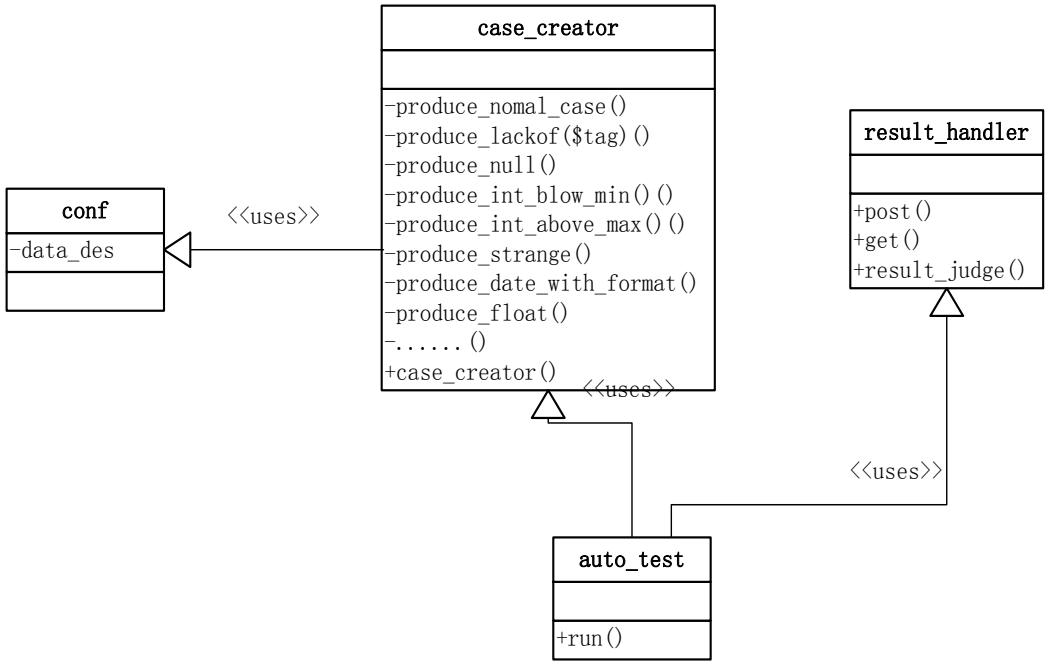


图表 1 参数类型和测试用例对应关系图

因此，基于上述规律，对参数的有效性的测试用例的设计，有很大的自动化空间，可以节省人力、缩短测试时间。

二、设计框架介绍

2.1 UML 图



图表 2 参数有效性检查自动化设计 UML 图

图 2 是自动化代码的框架。情况如下：

1、 conf: 里面是待测接口的参数具体情况，包括：参数名、是否必填、是否检查、类型、最大长度、最小长度（或者最大值、最小值）、可否为火星文、如果是时间还需要带格式。

2、 case_creator：它是核心部分。其中，以 produce 开头的都是生成具体的一个测试用例数据。produce_nomal_case 是生成一个标准的、合法的数据组，这个需要人工事先填好。其他的 produce_XXX 是在合法的数据组里面把某个字段改成对应的非法的。在这个类里，供使用者调用的，也是核心接口是 case_creator。

这个接口的功能如下：

- 读入 conf 文件
- 一次对每个参数进行 case 生成，例如：int 型，调用特定的几个 produce_XXX
- 对生成的每个数据组都设置好对应的错误码，expected_result。
- 将生成的<测试用例，期待结果>都放入 array，返回这个 array。

返回的 array 具体格式如下：

Args_name1 => 0 => <测试用例，期待结果>

=>1 => <测试用例，期待结果>

=>…..

Args_name2 =>0 => <测试用例，期待结果>

=>1 => <测试用例，期待结果>

=>…..

…..

3、 result_handler 类包括三个函数：

get : 把参数 get 出去

post: 把参数 post 出去

judge: 把参数和期望值对比，相同返回 true,错误返回 false.

4、 auto_test 类是控制中心。里面只有一个函数 run。这个函数的作用是：

1、 调用 case_creator 来生成数据。

2、 将生成的测试用例依次取出来，调用 result_handler 对应的方式发送测试数据并取得结果。

3、 最后调用 result_handler 的 judge 进行对比，并以自己想要的统计方式打印出来。

2.2 使用方法

基于上面的框架，对一个新网关接口进行参数有效性测试只需要进行一下几步：

1、 写 conf.

2、 把 case_creator 类里的 produce_nomal_case 的合法参数填写好。

3、 根据需要修改 auto_test 里的结果输出方式.

4、 调用 auto_test 里的 run.

5、 查看结果并分析查找 bug.

6、 人工手动补充些 case

三、结论

3.1 适用范围

由于此次自动化是建立在具有 1.1 中谈到的参数相互独立的接口上进行的自动化，对不具有这个特征的接口并不一定适用，如某些 server 的接口，server 接口由于内部逻辑复杂，很可能参数的取值范围是相互影响的，所以此次自动化并不适用。

简而言之，适用范围是参数相互之间具有独立性的接口。

另外，对以下几种参数也没有自动生成 case，需要最后人工补充：

1、 某个参数具有固定格式，如 query 接口的 order 参数：银行 id|order_id|date。这个参数格式复杂，自动化的代价太大，这个参数检查可归为人工手动补充 case。

2、 Sign，可以自动化，但还没有想到以什么方式收录到 case_creator 中来。

3.2 设计框架的优缺点

3.3 自动化程度

以网关接口为分析对象，网关参数越多，自动化生成 case 所占的比例就会越大，自动化程度越高。

以支付网关为例：

1、 charge 接口，一共 12 个 case，10 个是参数校验。自动化之前要设计 12 组数据，也就是 12 个 url；自动化之后，填写一个 conf 文件，设计 1 个正常数据，自动化 case 占比例 80%。

2、 charge_notify 接口，一共 70 个 case, 60 个参数校验。自动化之前设计 70 组数据；自动化之后，设计 11 组，自动化比例占 85%。

3、 query 接口，由于 order 格式复杂，自动化的框架不适用。

4、 refund 接口：一共 15 个 case，10 个参数校验。自动化程度 60%

5、 refund_notify 接口：43 个 case，30 个参数校验。由于里面有个参数格式复杂，不适合自动化框架，自动化 case 为 24 个，自动化程度 50%。

Charge	Charge_notify	Query	Refund	Refund_notify
80%	85%	0	60%	50%

从时间上看，支付网关和慢充网关对比如下：

网关名	接口数量	接口参数量	时间
慢充网关	3（全手动）	平均 10 个参数	Excel_case 1 个工作日 数据准备 3 个工作日 执行调试 1 个工作日 Bug 确认，收尾 1 个工作日
支付网关	5（其中自动化 4 个）	平均 12 个参数	Excel_case 1 个工作日 数据准备 0.5 个工作日 执行调试 1 个工作日 Bug 确认，收尾 1 个工作日

由两种对比数据可以看出，自动化程度一般情况下在 50%以上和节约了大量的数据准备的时间。

四、展望

4. 1 可推广之处

此次自动化虽然是针对网关的接口进行设计，仅适用于网关。但是对于参数的有效性是相互独立的其他类型的接口来说也一样适用，如部分 server 接口。

4.2 可改进之处

对参数的有效性检查的核心类 case_creator 应该还可以收录更多的格式复杂的 case，让自动化更彻底一些，只是现在还没有想到一个通用的方法和规律。

浅谈开源函数集的自动化解决方案

——以 Quercus 测试为例

作者：斯 如

选取优秀的开源框架，结合自己的业务场景，整合自己的业务框架，快速产出自己的应用，已是一种普遍的应用开发模式。但是，并非所有的开源框架都有一套完整的测试体系，保证对外提供的函数（或方法）没有缺陷，加上开源框架和自己应用整合的质量保证，我们常需要对开源框架在自己应用中的函数集，进行回归验收测试，且还有应用/或开源框架升级，持续回归的需求。

对于该类应用的开源框架测试，因其函数量大、规则不清晰，加上若项目周期短，常令人比较“纠结”。本文旨在以 Quercus 测试为例，介绍一种快速构建该类自动化测试集的方法。

一、背景分析

Quercus 是一种联合使用 Java™ 技术和 PHP 编写 Web 服务和应用程序的新方法。通过 Quercus 框架，可以将 Java 和 PHP 集成起来，从而支持将 Spring 和 Hibernate 等各种 Java 库集成到应用程序中。

被测应用，使用了 Quercus 框架作为应用引擎。链路为：应用提供 php 白名单->开发者 php 代码上传到指定地址->应用改造 Quercus 框架将 php 文件映射成 java 可执行文件->用户访问开发者 php 文件->应用环境运行对应 java 执行文件。

由此，用户访问开发者应用（php 文件），其中的每个函数能否被应用正确引擎，成为重要的测试范围。同时，考虑到应用快速迭代升级的需求，我们对 php 白名单函数快速产出自动化测试集，并在应用引擎接口测试层面和在集成环境中访问 php 层面进行回归测试。

仔细查阅网上 php 函数实例，发现很多优秀的网站有实例，且访问 url 和返回结果集可标准化。由此，结合已有的自动化思路，可快速生成自动化测试集，思路如下介绍。

二、测试思路

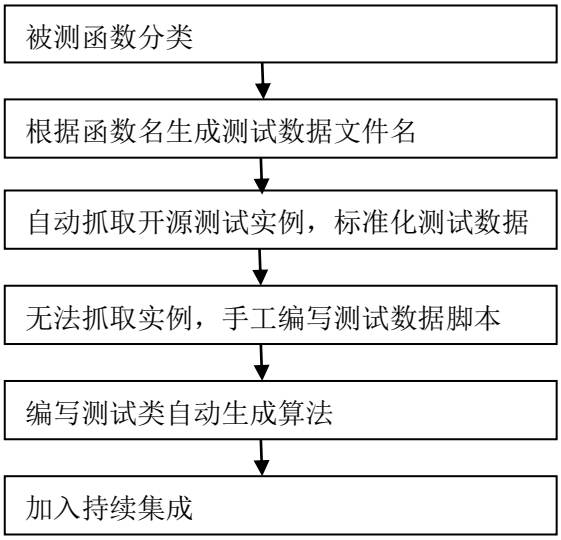
测试原理为：对支持的 php 函数白名单，使用页面自动化技术，访问并抓取已有网站的实例，生成 php 测试源数据；根据已标准化的测试源数据，自动化生成全部函数的接口测试类和自动化生成集成测试类。最后加入集成回归。

这里，接口测试和集成测试稍有差异，简述如下。

接口测试：

集成回归测试：

从 php 白名单到 java 测试类，步骤如下：



三、实例详解

1）被测函数分类

将 php 白名单源文件，备份在测试工程的_original（可自定义）目录中，并拆分成几个小文件（.txt），拆分原理可根据函数集本身划分，并以函数集命名文件名（如 array、string、date 等），方便管理。每个函数之间分割符建议一致，如换行符。

2）根据函数名生成测试数据文件名

这里，即读取_original 的.txt 文件，生成 php 文件名，具体可参见 java 转换和文件生成方法，其它场景可类比。

```
public class FileNameUtil {
    public static void creatFile(String path) throws IOException {
        File filename = new File(path);
        if (!filename.exists()) {
            filename.createNewFile();
}
    private static void doCreate(String oldFile, String newPath) {
        // 读文件
        BufferedReader reader = null;
        try {
            File file = new File(oldFile);
            reader = new BufferedReader(new FileReader(file));
            String tempString = null;
            int line = 1;
            // 一次读入一行，直到读入null为文件结束
            while ((tempString = reader.readLine()) != null) {
                // 显示行号
                System.out.println("line " + line + ": " + tempString);
                // 生成子文件夹(如string)和函数php文件名
                String path = newPath + "\\" + tempString.trim() + ".php";
                creatFile(path);
                line++;
            }
            reader.close();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (reader != null) {
                try {
                    reader.close();
                } catch (IOException e1) {
        }}}}
```

```
/**
     * 根据.txt生成.php
     */
    private static void generateCode() {
        // .txt路径
        String oPath = "quercus\\white\\_original";
        // .php文件路径
       String aPath = "quercus\\white\\";
        File file = new File(oPath);
        String test[] = file.list();
        for (int i = 0; i < test.length; i++) {
            System.out.println(test[i]);
            // 先创建文件夹
            if (!test[i].startsWith("_")) {
                String path = (aPath + test[i]);
                path = path.substring(0, path.length()-4);
                file = new File(path);
                file.mkdir();
                FileNameUtil.doCreate(oPath+"\\"+test[i], path);
            }
        }
    }
    public static void main(String[] args) {
        generateCode();
    }
}
```

3） 自动抓取开源测试实例，标准化测试数据

这里，开源实例采用了 w3school 的函数例子，它规范，且有标准的调用方法和期望结果。

例如：根据 php 文件名，生成 w3school 的 url，使用 selenium 模拟访问，抓取例子，标准化写到 php 文件中；

自动抓取方案，可选用 java 的 HttpClient 模拟浏览器请求的方法，或其它页面自动化工具，如 selenium。前者相对简单，但因返回页面有其它元素，需进行

结果提取和各种转义符处理；后者简单，但速度较慢。由于这个步骤是一次性的，所以选用了后者。

部分代码如下，调用 generateCode()即可。

```
public static void doSelenium(String url, String fileName) {
    WebDriver driver = new FirefoxDriver();
    driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
    driver.get(url);
    try {
        List<WebElement> e2 = driver.findElements(By.tagName("pre"));
        if (e2 != null && e2.size() > 1) {
            File file = new File(fileName);
            System.out.println(file.getAbsolutePath());
            // 写脚本
            String data = e2.get(1).getText();
            String data2 = data.trim().substring(0, data.length() - 3);
            FileUtils.writeStringToFile(file, data2, false);
            // 写期望结果
            List<String> result = new ArrayList<String>();
            if (e2.size() > 2) {
                String expected = "print \"预期结果:\".\""
                        + e2.get(2).getText() + "\";";
                result.add("");
                result.add(expected);
            } else
                System.out.println(url + "无期望结果");
            result.add("?>");
            FileUtils.writeLines(file, result, true);
        } else {
            System.out.println(url + "不存在");
        }
    } catch (NoSuchElementException e) {
        System.out.println("url 不存在");
        System.out.println(url);
    } catch (IOException e) {
        e.printStackTrace();
    }
    driver.quit();
}
```

```
/**
 * 根据.txt 生成.php
 */
private static void generateCode() {
    // .txt 路径
    String aPath = " quercus\\white\\";
    String url = "http://www.w3school.com.cn/php/func_";

    File file = new File(aPath);
    String a[] = file.list();
    for (int i = 0; i < a.length; i++) {
        // System.out.println(a[i]);
        String temp = a[i];
        if (temp.equals("simplexml")) {
            String path = aPath + temp;
            File file2 = new File(path);
            String test[] = file2.list();
            for (int j = 0; j < test.length; j++) {
                String pName = test[j];
                String fName = pName.substring(0, pName.length() - 4);
                if (!fName.startsWith(temp)) {
                    String curl = url + temp + "_" + fName + ".asp";
                    doSelenium(curl, path + "\\" + pName);
                } else {
                    String curl = url + fName + ".asp";
                    doSelenium(curl, path + "\\" + pName);
                }
            }
        }
    }
}
```

生成前后文件如下。

W3school 源文件核心部分

（如 http://www.w3school.com.cn/php/func_string_substr.asp）：

例子 1

```
<?php
echo substr("Hello world!",6);
```

输出：world!

生成 php 测试数据（string_substr.php）：

```
<?php
echo substr("Hello world!",6);
```

4） 无法抓取实例，手工编写测试数据脚本

Php 文件规范同 3）

5） 编写测试类自动生成算法

根据php文件名，自动生成java测试类，可先手动编写一个测试类（如a.java），其他类使用改动部分替代生成方法。改动点包括：文件名、类名、被测php文件名、包名等。

部分方法如下：

```
private static String upperName(String name) {
        StringBuffer result = new StringBuffer();
        if (name != null && name.length() > 0) {
            result.append(name.substring(0, 1).toUpperCase());
            for (int i = 1; i < name.length(); i++) {
                String s = name.substring(i, i + 1);
                if (s.equals("_")) {
                    result.append(String.valueOf(name.charAt(i + 1))
                            .toUpperCase());
                    i = i + 1;
                } else {
                    result.append(s);
                }
            }
        }
        return result.toString();
    }
    public static void doRepalce(String oldFile, String newFile,
            String newName, String phpName,String packageName) {
        List<String> result = new ArrayList<String>();
```

```
            //  读文件
            BufferedReader reader = null;
            try {
                File file = new File(oldFile);
                reader = new BufferedReader(new FileReader(file));
                String tempString = null;
                int line = 1;
                // 一次读入一行，直到读入null为文件结束
                while ((tempString = reader.readLine()) != null) {
                // 显示行号
                System.out.println("line " + line + ": " + tempString);
                    if (tempString.equals("package quercus.white.array;")) {
                    result.add("package quercus.white."+packageName+";");}
                    else if (tempString
                            .equals("public class ArrayDiffAssocTest extends
AbstractBaseEngineTest {")) {
                        result.add("public class " + newName
                                + " extends AbstractBaseEngineTest {");
                    } else if (tempString.contains("/white/array/array_diff_assoc.php")) {
                        result.add(tempString.replace("array/array_diff_assoc",
packageName+"/"+phpName));
                    }
                    else{
                        result.add(tempString);
                        }
                        line++;
                     }
                     reader.close();
                }
               catch (IOException e) {
               e.printStackTrace();} finally {
               }
            }
```

```
if (reader != null) {
                        try {
                              reader.close();
                        } catch (IOException e1) {
                        }
                  }
// 写文件
            File resultFile = new File(newFile);
            try {
                  FileUtils.writeLines(resultFile, result);
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }
      private static void generateCode() {
            String ePath = "quercus\\white\\";
            String aPath = "quercus\\white\\";
            String filename = "a.java";
            File file = new File(aPath);
            String test[] = file.list();
            for (int i = 0; i < test.length; i++) {
                  System.out.println(test[i]);// 先创建文件夹
                  if (!test[i].startsWith("_")) {
                        String path = (ePath + test[i]);
                        file = new File(path);
                        file.mkdir();// 生成文件
File file2 = new File(aPath+test[i]);
                        String test2[] = file2.list();
                        for (int j = 0; j < test2.length; j++) {
                        System.out.println(test2[j]);
                        String phpName = test2[j].split(".php")[0];
                        String newName = upperName(phpName) +"Test";
                        String expected = newName + ".java";
                        System.out.println(expected);
FileAccessUtil2.doRepalce(ePath + filename, path +"\\"+ expected,
                                    newName, phpName,test[i]);
                     }
                  }
               }
            }
```

```
public static void main(String[] args) {
    generateCode();
}
```

如最后生成的SubstrTest：

```
public class SubstrTest extends AbstractBaseEngineTest {
    String fileName = "/white/string/substr.php";
    @Test
    public void test_smoke() throws Exception {
        String result = testQuercus(fileName);
        String[] expected = result.split("预期结果:");
        super.doAssert(expected[0], expected[1]);
    }
}
```

其中，testQuercus 对于接口测试和集成测试调用方法不同，接口测试调用被测应用引擎方法，集成测试直接 httpClient 模拟访问。Assert 校验建议删除转义符，如"
|
|
|\\s*|\t|\r|\n"等。具体方法这里不再累述。

6） 加入持续集成

持续集成，可选 jenkins 或其它已有的持续集成回归体系。

四、总结

本次实战中，使用了 Selenium 框架，抓取优秀网站测试实例的自动化方法。它大大提高了测试效率，也可适用于其它很多场景。其中，标准化的测试数据文件和测试类，也可大大降低我们的维护成本，提高可读性。针对不同业务场景，接口测试和集成测试各司其职，亦成体系，纳入持续集成，是质量保证的重要基础设施。

抽奖性能测试设计

作者：雨霞

性能测试，是指对系统吞吐量的压测，它可以在系统发布到生成环境前提前暴露并解决性能瓶颈，它对系统稳定性至关重要。而性能测试的压测结果是否正确，是否更为真实的反应生产环境的情况，在很大程度上取决于测试方案的设计是否合理。

本文通过一个简单的模拟抽奖项目对性能测试方案的设计进行阐述，主要涉及测试需求、测试场景和监控系统与指标三个方面。

一、测试需求的剖析

测试需求包括明确测试范围（性能点）和期望值，性能点是指测试哪些页面或者接口，期望值是由 PV 与 TPS 转换模型获取期望高峰 TPS 值，如高峰 TPS=PV*2/(8*3600)，这样在测试时就有了是否通过的测试依据。

为明确测试需求，我们需要了解项目系统设计，甚至具体的业务逻辑。

模拟抽奖系统的网络拓扑图如图 1-1 所示。

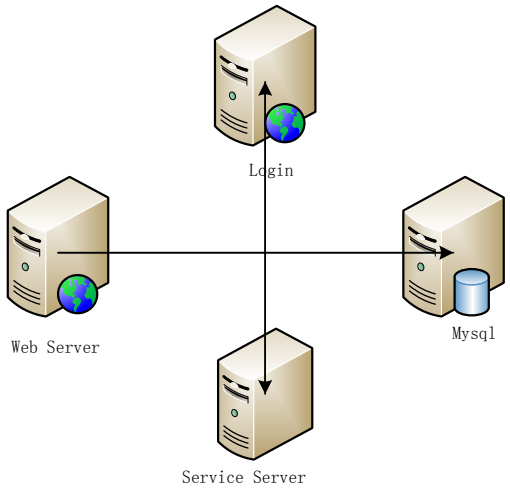


图 1-1 网络拓扑图

假设系统的主要业务需求如表 1-1 所示。

表 1-1

序号	概述	描述
1	抽奖页面	1、访问页面已登录用户跳转到抽奖页面，否则跳转到登录页面 2、页面展示抽奖按钮
2	卖家与活动	一个卖家最多发布 1 个活动，每个活动设置 50 张淘宝彩票，且活动在 2014-01-01 内有效。
3	买家中奖限制	1、中奖用户必须为女性，且中奖概率为 50% 2、每个买家对一个活动只能中一次奖

Web Server 代表抽奖页面，处理 http 请求，主要息是校验是否登录和展示抽奖信息。Service Server 代表抽奖和发奖逻辑，抽奖逻辑依次为用户合法性、活动有效性、中奖概率的校验，校验成功后更新活动表库存奖品数量，然后将卖家买家活动中奖关系插入中奖关系表；而发奖逻辑采用异步方式发送彩票号码到用户中奖详细记录中。 Mysql 主要是对用户信息、抽奖活动、中奖信息、发奖信息等数据的存储，当然也可以使用 oceanbase、cache 、文件等存储。

对于抽奖系统来说，它还是有一定特殊性的，除了保证系统吞吐量外，还需要保证系统不存在超卖和超中现象。为更好地理解超卖和超中的概念，我们将奖项的设置者视作卖家 seller，抽奖用户视作买家 buyer。在抽奖活动中卖家都会对固定奖项设置“库存”数量，超卖是代表所有买家抽中或系统发出的某一奖品的数量超过了卖家设置的数量，即卖家超卖。针对一次抽奖活动中卖家一般都会定义买家对固定的奖品只能中多少次，而超中是指买家针对某一种奖项其中奖次数超过了卖家设置的限制条件，即买家超中。

通过上面的简单了解，初步评估性能测试点为：抽奖模块的吞吐量、发奖模块吞吐量、中奖概率是否生效、超中及超卖是否存在。

二、测试场景的设计

测试场景根据性能点的覆盖程度分为单场景和混合场景，根据业务逻辑设计划分是相当复杂的，例如 cache 是否关闭，不同请求参数覆盖的业务逻辑等需视具体项目情况而定的。

在场景设计时，一定要明确该性能点的业务逻辑是怎样的，评估输入数据的不同对系统性能造成的影响，是否有必要构造不同的请求进行测试。另外，还需要要估算 DB 中的数据量和数据分布情况，测试数据的构造对系统的吞吐量也是至关重要的。

项目的测试场景可简单设计如表 2-1 所示。

表 2-1

序号	性能点	性能场景	期望TPS	期望RT	执行策略	备注
1	超中及超卖是否存在	买家超中	N/A	N/A	采用一个用户对大量的卖家活动进行抽奖，校验用户对每个活动的中奖数量是否为1？	单场景
		卖家超卖	N/A	N/A	采用大量女性buyer对同一卖家活动进行抽奖，校验用户中奖总记录是否超过活动奖品总量50？	单场景
2	中奖概率是否生效	中奖概率是否生效	N/A	N/A	抽奖用户为女性buyer对大量买家活动进行抽奖，在卖家奖品未有抽完的情况下，对中奖数量和抽奖请求数量进行校验，是否为50%？	单场景
3	抽奖模块的吞吐量	用户抽奖，关闭发奖逻辑	N/A	N/A	为最大化测试抽奖逻辑，用户为女性buyer对大量买家活动进行抽奖	单场景
4	发奖模块吞吐量	单独触发发奖逻辑	N/A	N/A	校验发奖的速度、发奖数量与中奖记录情况	单场景
5	抽奖与发奖吞吐量	混合场景	N/A	N/A	抽奖与发奖一起测试，即抽奖时发奖逻辑也打开	混合场景

单就超中和超卖现象，如何测试呢？这就需要我们设计测试场景。超中的极端测试：同一个用户在同一时刻针对同一卖家的某一个奖项进行抽奖，校验中奖情况是否超过了卖家针对同一个用户设置的数量；超卖极端测试：大量不同用户在同一时刻对一个卖家的某一个奖项进行抽奖，校验用户整体中奖数量和卖家发出的奖品数量是否相等，且需要校验是否超过了卖家设置的总“库存“。另外，在抽奖模块吞吐量测试时，也可以对用户中奖数量和卖家奖品发放数量进行校验，以验证在高并发情况下，抽奖模块是否正常的。

在执行测试时，有时为了更快速地响应业务测试需求，首先我们可以先执行混合场景的测试，若测试发现异常，再单个场景隔离开来进行测试。这样做有助于工作效率的提升，节约时间并且解放人力成本等。

三、监控系统与指标的确定

监控指标，画出系统网络拓扑图，确定各层面需要监控的性能指标。一般而言，监控指标包括TPS、RT、cpu、load、网络I/O、memory、JVM、缓存命中率、Exception等。

应用端性能指标，如表3-1所示。

表3-1

测试场景	前置条件	TPS	RT	server Resource	备注

				CPU	Load	网络 I/O	Memory	JVM	
性能点 1									CPU: <75% 和 Load:<4(4 核) 和 Java Memory:<80%
性能点 2									CPU: <75% 和 Load:<4(4 核) 和 Java Memory:<80%

数据库端性能指标，如表 3-2 所示。

表 3-2

数据库 IP	CPU	load	memory	I/O	SQL 平均执行时间	每秒完成事务数量	SQL 性能概析

在测试过程中，除了需要关注应用端的性能表现外，还需要关注数据库端的情况，并且要评估 db 的 QPS 和 TPS 容量，因为多数情况下应用端的机器数量是多于 DB 的机器数量的，当然除了 DB 端也还需要关注其他的系统情况，如登录应用 Login、cache 等的容量。

在做完方案设计后，需要召集项目相关的开发人员、PE、DBA 等成员通过会议进行测试方案的评审。当然，如果项目比较简单，彼此比较熟悉了，可以通过邮件或者电话的方式就测试方案沟通到位也是可以的。

综上，一个性能测试项目方案设计算是完成了，接下来就是准备环境、数据和脚本，准备工作完成后便可以测试了，根据测试结果和期望值进行结果分析，若不满足期望需要对系统进行调优，最后发布到生产环境。测试人员需要对项目的产品要有 owner 意识，定期对系统进行 check 和总结。监控一旦有异常情况发生，随时随地进行问题的排查与跟进。其实性能测试流程走通还是蛮容易的，但要全面的进行方案设计、深入的测试和调优，就需要一定的技术和项目经验了。

搭建嵌入式产品自动化测试框架全过程

作者：妞妞

摘要:嵌入式产品(Android 系统)实现自动化测试的全过程,从零基础开始,分三个阶段讲解搭建自动化测试架构。第一阶段讲述了基础知识例如 python 语言，Monkey runner 学习的重点,目标是达到产生一个简单脚本的程度。第二阶段说明如何搭建一个产品的自动化测试框架。第三阶段说明通过实践的总结，对搭建的框架进行改善。此文章是经过两年的实践总结出来的，具有很强的现实意义。此文档主要讲解学习思路和学习方法,细节的内容只会给予学习方法的说明。

关键字：自动化测试，Android ， 完整解决方案，零基础，实践

前言：

从事软件测试近 10 年，经常听到测试人员对于重复机械化测试的抱怨，对于重复机械化的测试会让测试人员看不到自己的前景,进而导致人员离职率较高,渐渐的已经变为一个恶性的循环。

如何解决这个问题，一直困扰着我，我于 2012 年开始致力于研究基于 Android 自动化测试，虽然网上可以找到很多有关自动化测试的资料，但是很少有一个完整的从零开始的解决方案，截至到目前已经积累了一些经验，现在我将经验整理，产生一个完整的解决方案，希望可以帮到广大的有相同困惑并有想法改变现状的网友们，目前还处于实践的初级阶段，后续会在实践过程中继续总结经验，持续进行改进。

我相信：用伟大的方法去做小事情，小事情也会变成伟大的事情！软件测试的兄弟姐妹们，一起冲啊!!

正文：

第一阶段：基础知识学习

1、配置开发环境安装说明

1）需要下载的工具：

下载 JDK(免费)

下载 Android SDK(免费)

下载 Python(免费)

2）需要配置环境：

在 Windows 的环境下,系统属性->高级->环境变量:添加两个文件夹的路径:JDK 的 BIN 文件夹路径和 SDK 中 Platforms tools 的路径两个路径之间使用“;”隔开。

3）Android 内部配置

将 Android SDK 中 Platform tools 文件夹中的 adb.exe 放到 Tools 文件夹中

2、Python 语言的学习重点

1）数据类型

a) 链表的基本概念和一些相关函数的使用(实现脚本框架的模块化时会使用到此部分知识

b) 字典的基本概念，关键是如何使用字典中定义的元素

2）语句

a) For 语句

在编写压力测试的脚本时会使用到此语句

b) While 语句

在编写压力测试的脚本时会使用到此语句

c) If else 语句

在编写需要用到判断的语句，例如在不同的条件下输出不同的测试报告时会用到此语句

3）String

a) 了解字串的使用方法，例如引号的使用，如何将字串串联起来。

b) 了解对字串做处理的一些函数的意义和用法

String 会在输出测试报告，自动建立照片所在文件夹和自动建立照片名称时使用

4）函数的调用和定义的方法

a) 函数中参数的定义方法

b) 调用函数的方法以及在调用函数时参数的传递方法

对于自动化脚本模块化时，此部分的知识会用到

5）异常的处理

对于一些已知的有可能产生的错误进行预先处理

6）导入模块的方法

a) 系统自带的模块

例如 sys 模块：将各个模块的路径放到环境变量中，以保证各个模块的脚本可以正常运行 os 模块：自动生成拍照的照片和测试报告的路径

b) 用户自己产生模块的产生和调用的方法在第三章和第四章中会用到此部分内容

7）File 的处理

a) 如何打开一个文件

b) 如何在文件中写入相关内容

3、Monkey runner 工具的学习重点

1) Monkeyrunner 类

此类中的函数主要用于连接设备，例如

Device=MonkeyRunner.**waitForConnection()**(此时与 PC 连接的设备开始尝试与 monkeyrunner 工具进行连接)

2) MonkeyDevice 类

此类中的函数主要用于激活活动，发送键盘事件，touch 事件，drag 事件，press 事件等。例如 device.press('KEYCODE_DPAD_RIGHT','DOWN_AND_UP') 就激活了向右移动的一个事件，需要将此类中的所有事件均做简单了解，因为我们需要在手机上模拟用户做的所有操作，都在此类中做定义。

3) MonkeyImage 类

此类中的函数用于屏幕抓图，产生图片，对比图片，产生报告。例如 device.takeSnapshot() 此函数实现屏幕抓图。需要将此类上述 4 个方面的函数做简单了解，因为我们产生测试报告需要使用到上述 4 方面的函数。

4、第一个脚本

当学习了上述基本知识后可以产生如下脚本，实现的功能是新增一个电话本资料并保存(产生两个照片，根据两个照片是否一致来判断测试结果并产生 Log)

```
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage->即将 monkeyrunner 的模组导入到 python 中，这样才能在 Python 中使用 monkeyrunner3 个类中的函数。
```

```
deice=MonkeyRunner.waitForConnection()->将手机与 PC 连接在一起
#以下的步骤为产生第一张的图片
device.startActivity(component='com.android.contacts/.activities.PeopleActivity')->进入电话本的主菜单
MonkeyRunner.sleep(1)
device.startActivity(action='android.intent.action.INSERT',
component='com.android.contacts/com.android.contacts.activities.ContactEditorActivity')-> 进入新增电话本界面
MonkeyRunner.sleep(1)
device.type('basicfunctiontest')->输入电话本的姓名
MonkeyRunner.sleep(1)
device.press('KEYCODE_DPAD_DOWN','DOWN_AND_UP')->光标向下移动
MonkeyRunner.sleep(1)
device.type('10010')->输入电话本资料的号码
MonkeyRunner.sleep(1)
device.press('KEYCODE_DPAD_DOWN','DOWN_AND_UP')->光标向下移动
MonkeyRunner.sleep(1)
#press back key to save the item
device.press('KEYCODE_BACK','DOWN_AND_UP')->保存此笔资料
MonkeyRunner.sleep(2)
#take a photo
Frist=device.takeSnapshot()->在已保存界面拍照
MonkeyRunner.sleep(2)
First.writeToFile('d:\\Frist.png','png') ->将第一张图片保存到 d 盘中
MonkeyRunner.sleep(1)
device.press('KEYCODE_BACK','DOWN_AND_UP')->回到电话版主菜单界面
```

```
#以下步骤为产生第 2 张图片的过程
device.startActivity(action='android.intent.action.INSERT',
component='com.android.contacts/com.android.contacts.activities.ContactEditorActivity')-> 进入新增电话本界面
MonkeyRunner.sleep(1)
device.type('basicfunctiontest')->输入电话本的姓名
MonkeyRunner.sleep(1)
device.press('KEYCODE_DPAD_DOWN','DOWN_AND_UP')->光标向下移动
MonkeyRunner.sleep(1)
device.type('10010')->输入电话本资料的号码
MonkeyRunner.sleep(1)
device.press('KEYCODE_DPAD_DOWN','DOWN_AND_UP')->光标向下移动
MonkeyRunner.sleep(1)
#press back key to save the item
device.press('KEYCODE_BACK','DOWN_AND_UP')->保存此笔资料
MonkeyRunner.sleep(2)
Second=device.takeSnapshot()->在已保存界面拍照
MonkeyRunner.sleep(2)
Second.writeToFile('d:\\Second.png','png')->保存第 2 张图片
MonkeyRunner.sleep(1)
result=Second.sameAs(Frist,0.8)->对比两张照片
if result(对比的结果为 Pass 产生 log)
    f= open('d:\\pass.log','r+')->在 D 盘打开文件
f.write('the action adding item to phonebook were passed')->写入指定的 Log 内容
else: (对比的结果为 Fail 产生 log)
    f= open('d:\\fail.log','r+')->在 D 盘打开文件
f.write('the action adding item to phonebook were failed')->写入指定的 Log 内容
```

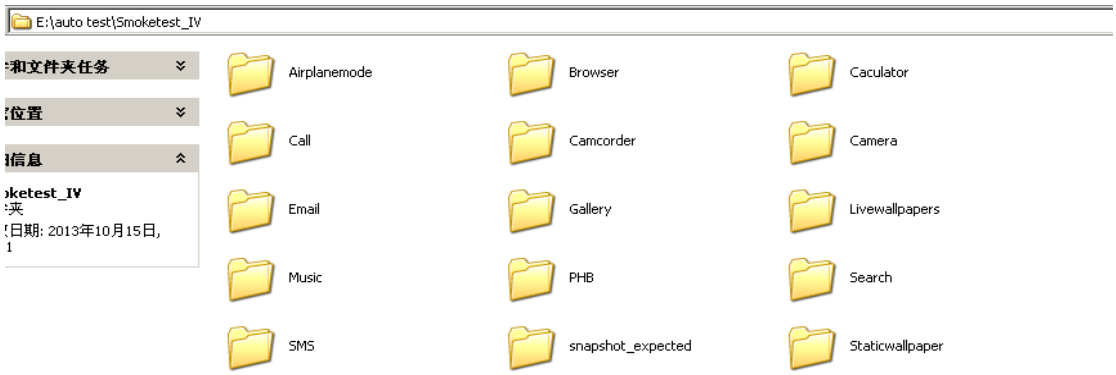
第二阶段：自动化框架的搭建

1、目标

将不同的功能作为独立的模块进行被调用和维护，方便维护，同时也可以使自动化脚本的框架清晰明了

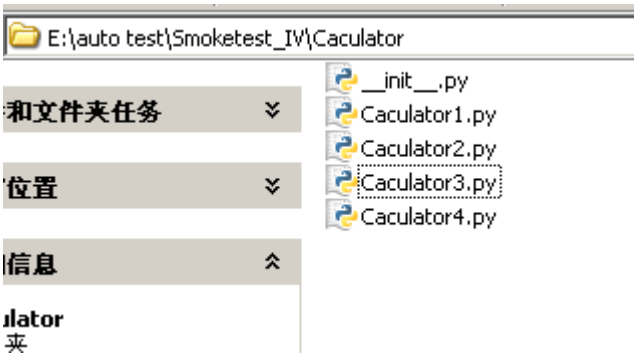
2、文件夹建立

在某一存储位置下(例如:E 盘)建立一个文件夹：例如 auto test\Smoketest_IV。在此文件夹下建立有关各个模块的子文件夹(有多少模块就有多少子文件夹)：例如 calculator 和 PHB 就必须分别建立两个文件夹，如【图 1】



【图 1】

如果 calculator 其中有两个 item 则在 calculator 中建立四个".py"文档一个命名为 calculator1.py,calculator2.py,calculator3.py, calculator4.py。如【图 2】



【图 2】

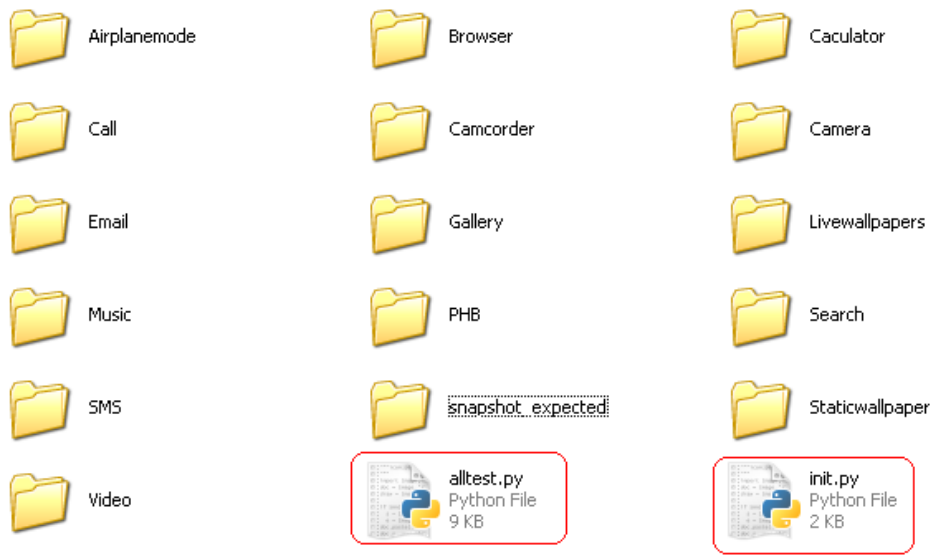
注意： 每个文件夹中必须有一个名称为"__init__.py"的内容为空白 py 文档,只有包含此文档则此文件夹中的模块才能被其它的脚本正常调用。

Autotest 此文件夹中放入调用各个脚本的 alltest.py 和其中含有各个模块都会调用的函数的 init.py 此两个脚本。

注意：每个模块只能实现一个大的功能模块，如此功能模块是由两个脚本组成：例如 calculator 由 4 个 item 组成，则在此 4 个模块需要分成 4 个".py"文档。

3、脚本的分配

主要分成 3 个方面：alltest.py initmodule.py 和其它各个功能的子模块(如下图中的各个文件夹)如【图 3】



【图 3】

1) alltest.py

此脚本的意义是调用各个模块中函数并执行，即执行自动化测试时只要运行此脚本即可，因为它的用法就是调用各个模块并执行各个模块。

如【图 4】

```
import sys
import os
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage
sys.path.append('d:/autotest')
device=MonkeyRunner.waitForConnection()
try:
    from caculator import caculator1,caculator2
except ImportError:
    print ('The path is wrong!!')

import init
caculator1.caculator1(device)
caculator2.caculator2(device)
```

【图 4】

alltest.py 脚本的详细说明：

from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage ->将 monkerunner 的类导入此脚本中

import sys ->导入 Python 自带的类，此类中有我们需要的 append 此函数

sys.path.append('d:/autotest') ->即指向各个脚本所在的文件夹

device=MonkeyRunner.waitForConnection()->将手机与 PC 进行连接

try:

from calculator import calculator1,calculator2

except ImportError:

print ('The path is wrong!!!')->上述脚本的意义是从 calculator 此文件夹中导入 calculator1 和 calculator2 和从 PHB 此文件夹中导入 phb1 和 phb2。

注意：导入模块，此时模块是不会被执行的，如果希望被执行需要调用此模块中定义的函数。使用 try…except 的目的是：如果发生导入失败的现象会告诉用户错误是什么，也可以藉由此函数当发送 except 中的错误是可以将错误跳过去，继续执行下面的脚本。

import init->将 init 此脚本导入到 alltest.py 此脚本中 calculator1.calculator1(device)calculator2.calculator2(device)->从 calculator1 此模块中调用 calculator1(device)此函数。

注意：此处才会真正执行 calculator 此函数中定义的实质的内容。

2) init.py

此脚本的意义是抽象各个模块都会使用的一些功能，例如这里抽象出 wirtelog 此功能，每个模块测试完毕后都需要根据测试结果产生 log（即测试结果）。如【图 5】（当然随着脚本开发的深入，你可以抽象更多的共用的功能放入此脚本中，例如拍照，对比等都可以抽象到此脚本中）

```
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage

def writelog(result,testpicture,formalpicture,file,passcontent,failcontent):
    result=testpicture.sameAs(formalpicture,0.8)
    MonkeyRunner.sleep(1)
    if result:
        file= open('d:\\pass.log','a')
        file.write(passcontent)
    else:
        file= open('d:\\fail.log','a')
        file.write(failcontent)
```

【图 5】

initmodule.py 脚本的详细说明如下

from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage
->将 monkerunner 的类导入此脚本中
Def writelog(result,testpicture,formalpicture,file,passcontent,failcontent):->定义一个输出测试结果的函数，其中 Result 代表->test 与 formal 的图片对比结果
Testpicture->test 产生的 pictureFormalpicture->formal 产生的 picture
File->此脚本输出结果的文档的变量名称 Passcontent->如此脚本运行结果是正常的，则产生的输出结果 Failcontent->如此脚本运行结果是异常的，则产生的输出结果
result=testpicture.sameAs(formalpicture,0.8)->对比 test 和 formal 的图片
if result:file= open('d:\\pass.log','a')file.write(passcontent)->如果结果是 true，则以追加的方式打开文档(即不会文档中已经存在内容，只是会不断的向下累加内容)
失败的状况不再讲解，同 pass 的状况。

3) 各个子模块

这些脚本的作用是：具体操作手机各个功能。例如【图 6】是对手机中的计算器进行加法功能的测试脚本。这些脚本实现的相对独立的功能定义为函数，以便 Alltest.py 此脚本调用并运行。

```
from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage
import init
passcontent='the Calculator item1 : Addition calculation was passed!\n'
failcontent='the Calculator item1 : Addition calculation was failed!\n'
def caculator1(device):
    device.startActivity(component='com.android.calculator2/.Calculator')
    MonkeyRunner.sleep(2)
      # Input number of addition
    device.type('4566+9888')
    MonkeyRunner.sleep(1)
    device.type('=')
    MonkeyRunner.sleep(2)
      #take a photo for the result
    formalCalculatoritem1=device.takeSnapshot()
    MonkeyRunner.sleep(1)
    formalCalculatoritem1.writeToFile('d:\\formalitem1.png','png')
    MonkeyRunner.sleep(1)
    device.press('KEYCODE_BACK','DOWN_AND_UP')
    MonkeyRunner.sleep(2)

      #generate the test picture
    device.startActivity(component='com.android.calculator2/.Calculator')
    MonkeyRunner.sleep(1)
      # Input number of addition
    device.type('4566+9888')
    MonkeyRunner.sleep(1)
    device.type('=')
    MonkeyRunner.sleep(2)
      #take a photo for the result
    testCalculatoritem1=device.takeSnapshot()
    MonkeyRunner.sleep(1)
    testCalculatoritem1.writeToFile('d:\\testitem1.png','png')
    MonkeyRunner.sleep(2)
    resultCalculatoritem1=testCalculatoritem1.sameAs(formalCalculatoritem1,0.8)
    MonkeyRunner.sleep(1)
    init.writelog(resultCalculatoritem1,testCalculatoritem1,formalCalculatoritem1,file,passcontent,failcontent)
    device.press('KEYCODE_BACK','DOWN_AND_UP')
    MonkeyRunner.sleep(1)
    device.press('KEYCODE_BACK','DOWN_AND_UP')
    MonkeyRunner.sleep(1)
```

【图 6】

Calculator 脚本的详细说明：

from com.android.monkeyrunner import MonkeyRunner,MonkeyDevice,MonkeyImage ->将 monkerunner 的类导入此脚本中
import initmodule->将 initmodule 模块导入到当前的模块 passcontent='the Calculator item1 : Addition calculation was passed!\n'
failcontent='the Calculator item1: Addition calculation was failed!\n'
->将此模块执行完毕后，产生 log 的内容定义出来 def module'sname(device): (例如:def calculator1(device):)->将此脚本的主要实现的功能定义为一个函数
resultCalculatoritem1=testCalculatoritem1.sameAs(formalCalculatoritem1,0.8)->避免产生在调用 init 模块中的 writelog 函数时发生错误，需要先执行此语句
init.writelog(resultCalculatoritem1,testCalculatoritem1,formalCalculatoritem1,file,passcontent,failcontent)->调用 init 中的共用函数 wirtelog
将每个脚本都会固定使用的小功能模块：将脚本的测试结果打印到某一个文档中，并输出抽象为一个模块，为每个脚本所调用。

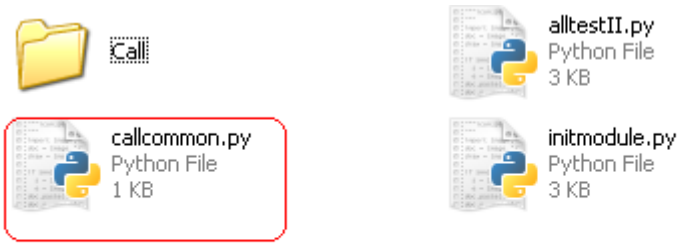
第三阶段：优化框架

1、当前框架存在问题

在写脚本时发现较多的功能是共用的，如果能将这些功能抽象出来，就能提高代码的复用性。

2、文件夹的改进

与之前比多了一个脚本 callcommon.py 这个脚本即为将 call 此功能中共用的功能抽象出来，供其他模块调用。如【图 4】



【图 4】

3、脚本的改进

多了一个 functioncommon.py 此脚本，此脚本中包含了所有的抽象函数，为了后续的方便调用，函数的作用要单一，不能几个功能集中在一个函数中。

Functioncommon.py（例如 callcommon.py）此脚本中定义的函数均需要使用形式参数 Function.py(例如 call.py)此脚本中需要调用 functioncommon.py 此脚本中的函数需要注意：参数传递问题->总体的原则是在 function.py 中的函数在调用 functioncommon.py 中的函数时，函数中所使用的参数都是形式参数，只有在 Alltest.py 中调用 function.py 中的函数时，所使用的参数为实际参数，即在执行函数前，需要将所有参数进行实际的定义。

例如 【图 5】为定义的抽象函数脚本即 functioncommon.py，

【图 6】为需要调用抽象函数的功能模块即 function.py

```
def callcommon(device,target_Call1):
    runcomponent=target_Call1['package_name']+'/'+target_Call1['launch_activity']
    device.startActivity(component=runcomponent,data=target_Call1['data1'],action=target_Call1['action'])
    MonkeyRunner.sleep(10)
```

【图 5】

```
def Call1(device,target_Call1):
      #generat formal picture
      #dial out the number of 10010 and the system will connect automatic.
    testcase=__name__.encode('utf-8')
    callcommon.callcommon(device,target_Call1)

      #take a photo
    a=initmoduleIIimprovement.take_formal_snapshot(device,testcase,'moduleIIformal')
    MonkeyRunner.sleep(2)
      #end the call
    device.press('KEYCODE_ENDCALL','DOWN_AND_UP')
    MonkeyRunner.sleep(3)
```

【图 6】

目前我们的自动化还处于不断完善的阶段，也十分渴望能够得到专业人士的批评指正！

总结

通过与其它公司的测试 leader 沟通，发现大家都对自动化测试有一致的看法：看上去很美，实施起来却很难，因为维护成本太高。我的感觉，解决如下问题，自动化测试是可以很好的服务我们：

1、不能将开发自动化测试脚本的人员和实施手动测试人员分开只有手动测试

人员才能了解什么样的自动化才是我们真正需要的。且自动化脚本的使用者也是这些开发者，这样在后续的测试中，他会很敏感的发现问题，并可以很快的解决。有的公司是有专职的自动化测试团队，他是与手工测试人员分开的，因为缺乏有效的沟通和管理，开发出来的脚本往往缺少实用性，而且不断改善也缺乏动力。

更重要的是让手动测试人员有成就感，测试人员不再是没有技术含量，只是进行重复测试的测试机器了。

2、要有越挫越勇的精神

我们在刚开始开发脚本时，有一些测试人员反应，唉，这样的脚本没太大意义，还是手工测试效率好。但是经过两年的不断改进，这些脚本已经实实在在的给予我们很大的帮助，人力也降低了很多。现在我的团队已经没有人在质疑自动化的意义。反而软件工程师在做单元测试时，还要向我们要自动化脚本。

3、要有系统的眼光看待自动化测试

即必须将测试工具融入到测试策略和测试用例中。自动化不能与手动测试脱离，我们需要整理一份测试用例数据库，此数据库需要将测试用例分为两种：手工测试和自动化测试，这样 Leader 在安排工作时可以很明确的知道哪些测试用例是可以实现自动化的。且此数据库需要不断的更新，目标就是最大化的实现自动化。

4、一定要不断的完善自动化测试框架

不断的学习新的工具，新的语言，不断的尝试，不断的思考：这些工具和语言能为我做什么。那些抱怨测试没有前途的兄弟姐妹，想想你为改善工作做了什么努力？一个行业之所以成为一个行业一定有它的必要性，最近一直有一句话萦绕着我：用伟大的方法做小事情，小事情也会变成伟大的事。

谨以此话与大家共勉！

金融行业信息系统可靠性问题与建议

作者：赵亮

一、金融业信息系统可靠性事件频发

随着电子计算机和网络技术在计算机行业的开发和应用，金融信息系统已成为银行业务操作和管理现代化必不可少的工具，金融电子化极大地推动了集团财务公司、银行、证券公司、保险公司和非金融支付机构等金融业务的发展，这些机构金融业务的运转也越来越依赖于其信息系统，然而金融电子化的重要地位和它快速发展的同时，也衍生了许多风险，正在威胁着金融电子化的可靠性，严重影响了金融业务的开展。例如，近期发生的一些金融行业安全可靠性事件为我们敲响了警钟。

事件 1：工行系统故障

2013 年 6 月 23 日工行系统发生故障事件（以下简称“6·23 事件”），中国工商银行信息科技部就正式作出内部通报，这份通报称，工行数据中心（上海）主机系统出现故障，是由于 IBM 提供的主机 DB2V10 版本内存清理机制存在缺陷引发。通报也提及了一些管理问题，通报称：“（数据中心上海）没有按照‘第一时间恢复生产’的要求采取果断措施及时进行回退，并且回退过程不坚决，耗时较长。”。

事件 2：招行借记卡业务发生系统故障:

2013 年 5 月 12 日 招商银行业务系统瘫痪，中午的一两个小时里，几乎全国范围的招商银行 ATM 机、POS 机、网银甚至客服电话，都无法正常工作，原因是一台前置机发生故障。

事件 3：原平安银行系统发生故障

2011 年 9 月 20 日，平安银行曾出现罕见的外部电源整体断电故障，导致大部分银行系统出现瘫痪，断电导致系统故障时间近 24 小时，直至 21 日下午才恢复。

事件 4：中国银行系统

2012 年 12 月 15 日，中国银行采用的 IBM 大型机在运行过程中突然宕机，长达四个小时。

事件 5：日本东京证券交易所系统故障

2005 年 11 月 1 日，日本东京证券交易所股票系统发生大规模系统故障，导致所有股票全面告停，短短 2 小时造成了上千亿的损失。这次事故的原因是不久前为增强系统处理能力而更新的交易软件程序存在缺陷。虽难在工程师的紧急抢救下系统得以恢复，但这次事件已经在整个金融界留下挥之不去的阴影。

事件 6：美国纽约证券交易所系统故障

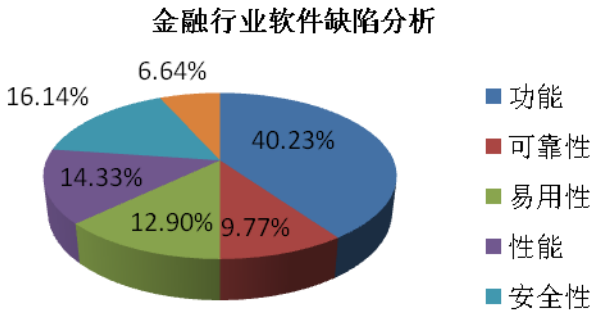
2010 年 10 月 13 日电 据香港《文汇报》综合报道，美国纽约证券交易所周一下午收市时出现问题，部分在主板市场及美国证券交易所市场挂牌的股票，过了收市时间 50 分钟后仍未停止交易。

有专家说，如果银行系统中断 1 小时，将直接影响该行的基本支付业务；中断 1 天，将对其声誉造成极大伤害；中断 2-3 天以上不能恢复，将直接危及其他银行乃至整个金融系统的稳定。而有调研机构对不同行业关键业务中断带来的损失进行过调查和统计：服务器宕机 1 分钟，平均会使银行业损失 27 万美元，证券业损失 45 万美元，这说明了关键业务平台对于稳定性和可靠性的要求。

二、主要问题及原因分析

上述频发的可靠性事件，不仅给金融服务机构本身造成了较大的经济损失和社会信誉损失，而且给广大用户带来了经济损失，给其今后业务的开展带来了不利的影响，影响了金融行业相关部门的业务连续性和相关业务开展，影响了金融稳定和安全，给金融行业发展带来了不利的影响。

根据中国软件评测中心发布的《2012 年中国软件产品质量年度报告》显示，在金融行业中软件系统的可靠性缺陷大约占到所有缺陷的 9.77%。



因此，金融信息系统的安全可靠性的缺陷目前日益成为阻碍金融行业业务发展的重要因素之一。中国软件评测中心对安全可靠性缺陷成因进行了研究，发现究其缺陷产生的原因，主要表现在管理和技术两个层面。

从管理层面来看，某些金融服务机构缺少对信息系统进行有效地系统运维管理和风险预防的手段和措施，无法当系统发生故障时进行及时、有效恢复，当可靠性风险发生时无法有效将风险的影响降到较低程度，主要问题表现在以下方面：

1、IT 运维管理重视程度不够

从目前金融服务机构 IT 运维水平来看，参差不齐，但综合水平不高，有些

机构只是重视了某一方面而忽略其他方面的内容。

1）缺少运维管理制度和操作规程

有些机构投入了大量资金为关键系统配置备份设备，并建立了同城异地灾备机房并实现了数据级备份和应用级备份的环境，但是并没有建立相关的管理制度及操作程序，因此在出现系统故障时没有一套程序去指导系统恢复延误系统恢复时间。在日常运维维护中，缺少设备管理、监控管理、变更管理的制度和操作规程，如在设备管理中缺少设备的操作规程、操作日志，在发生故障后运维人员无法及时进行回溯找到出现故障的原因，从而减少恢复周期；没有实行有效地对网络设备和服务器进行监控及其异常处理机制，当出现异常时无法实时监控到系统故障状态实现预警，能够自动保护当前所有状态。

2）制度和规程没有贯彻执行和落地

有些机构建立了相关管理制度及操作程序，但并没有在实际中执行下去，在一些需要有过程记录的地方，没有相关记录，制度和程序如同摆设，出现此原因，一是没有在制度和日常对运维人员考核中做出日常运维工作的规定，二是没有建立运维人员组织架构及其相关职责去执行，三是建立人员组织架构和职责，但没有实际人员去执行。

3）缺少应急预案的管理和演练

对于重要系统，需要建设完备的灾备体系，建立完善的应急预案 并且需要确保灾备和应急预案的有效性。然而，有些金融服务机构缺少针对不同事件的应急预案，对应急预案的启动条件、处理流程和系统恢复流程考虑不够周全，在实际发生故障时不能及时进行处理。

4）对 IT 运维人员的培训不够

在从事金融行业的 IT 运维人员，人员的技术水平肯定是参差不齐的，需要在故障发生时能够有技术能力和运维知识去进行处理，需要定期对相关人员根据岗位进行培训。但目前有些无法及时对故障进行及时处理的一根本原因是培训力度不够，导致在系统故障发生后未能及时机型恢复。

从技术层面来讲，某些金融服务机构对金融信息系统保护策略存在考虑不够充分，是导致系统频繁发生故障，系统可用性程度比较低，问题主要体现在以下方面：

1）缺少监控措施

监控措施主要包括对网络流量进行监控、对网络设备和服务器的运行状态进行实时监控，如：CPU、内存、磁盘 IO 等资源的使用情况设置阈值，当超过其

阈值时给予预警，并通知给运维人员进行相应处理。然而一些金融服务机构缺少这方面的控制措施，不能及时预防系统故障的发生。

2）全面的本地备份策略缺失

没有对实施全面的本地备份策略，如：重要的网络设备未有冗余和备份、互联网接入未做到双链路、服务器没有实现双机热备，当遇到突发故障时系统无法及时进行恢复，造成业务中断和业务数据的丢失。

3）缺乏有效的异地备份策略

缺乏有效的异地备份策略主要表现在灾备机房未做到应用级备份，在系统故障发生时，不能及时通过灾难备份系统进行恢复，造成业务中断时间过长。

4）没有完善的故障恢复策略

没有制定系统备份控制、系统备份制度、数据库备份恢复规范，规定备份系统自动执行恢复操作，造成系统恢复人员无法恢复系统时间过长甚至无法恢复业务数据。

5）未对非法数据进行有效性控制

非法数据指的是不满足类型、长度和格式的数据，这些非法数据导入到系统中可能会导致比较严重的问题，如某证券公司订单执行系统针对高频交易在市价委托时，对可用资金额度未能进行有效校验控制，导致特定情况下生成预期外的订单。由于订单生成系统存在的缺陷，导致在短短 2 秒内，瞬间重复生成 2 万多笔预期外的市价委托订单。

三、 金融行业可靠性提升建议

为促进金融行业业务的健康发展，就需要提高金融行业信息系统软件的质量，改进相关的管理措施。中国软件评测中心在信息系统软件质量提升方面做了很多工作。目前金融服务机构非常关注金融环境的优化，其信息系统的可靠性决定了金融环境优化是否取得成效的评判标准之一，因此信息系统的可靠性方面却来越关注，但系统可靠性的提升并不是一日之工，不是通过购买设备能够得到解决的，最根本的还是在技术和管理层面上。在本节中，将通过中国软件评测中心在金融行业系统可靠性测试方面丰富的经验，给出金融行业信息系统可靠性提升建议。

1、建立完善的备份策略、机制和制度

对于金融服务机构来说，业务涉及到大量资金交易，如何保障交易资金的安全，建立完善的数据备份策略和机制是非常有必要的，备份策略上来说应该实现对本地或异地的备份，本地备份应实现数据级的备份，异地至少应实现应用级的备份，如果能实现数据级的备份更好，能够有效地实现在出现故障时数据能够及

时恢复，业务数据能够在故障中不丢失，同时也应做到对关键的服务器、网络链路、互联网设备的备份和冗余，业务系统能够在故障中迅速恢复，保障业务连续性。

2、构建业务连续性管理制度

信息系统运维管理人员需要建立完善的备份恢复管理流程和制度，相应人员能够按期对数据进行备份，故障时按照流程迅速对系统进行恢复；需要对危机事件进行分类，设立危机处理程序来预防影响业务连续性的事件发生，能够在事件实际发生的时候进行及时有效处理；需要制定应急处理预案，为不同事件给出不同场景恢复预案，同时要求建立应用及恢复预案；需要形成数据备份和恢复管理制度，对备份的数据范围、频率做出规定，实现备份日志的记录。

3、定期应急恢复预案演练和培训

建议定期对运维人员按照岗位要求进行培训，能够每年实施应急演练，充分掌握当系统发生灾烂性故障时，能够按照符合金融行业规定的时间恢复系统。

4、实施完整有效的系统资源实时监控

构建系统实时监控系统，监控的第一个目标是网络流量、网络设备和服务器的资源占用情况以及服务器上运行的程序进行监控，设置相关阈值，超过阈值时进行报警通知相关运维人员。

5、加强系统可靠性检测

即使在可靠性方面都做了很多工作，但可能只是注重一方面而忽视了另一方面，做到全面是比较难的。因此建议可组建技术团队或委托专业软件测评机构对系统的可靠性方面进行测评。系统可靠性隐患是否能及时发现，并进行修复，因此，对系统进行定期的可靠性测评是非常必要的，能有效保障对系统的正常运行。

SOA 服务质量分级评价方法的研究

作者：王晓芹

所谓 SOA，是一种架构模型，它将应用程序的不同功能单元(服务)通过服务之间定义良好的接口和协议联系起来，其接口采用完全中立的方式来定义，独立于实现具体服务的硬件、操作系统和编程语言，这使得构建在各种这样系统中的服务可以以一种统一和通用的方式进行交互。基于 SOA 可以开发并集成业务中所应用到的不同组件。与面向对象开发、面向组件开发等传统方法相比，SOA 架构可以帮助客户提高业务流程的灵活性，确保 IT 系统快速、便利、经济地适应并支持不断变化的业务需求。增强企业底层的 IT 基础架构，并复用现有的 IT 投资等潜在优势。

SOA 架构以服务为中心。而服务是一种独立于应用及其运行计算平台的自包含的可复用的软件。SOA 服务关注于业务级任务、活动和交互。每种服务都能提供对该定义功能集的访问。它拥有定义良好的接口，能使业务任务和执行这些任务所需的确切 IT 组件之间形成一对一的映射。SOA 定义了系统由哪些服务组成，描述了服务之间的交互，并将服务映射到一个或多个具体的实现。这种将业务功能实现为服务的方法可以增强系统的灵活性，系统通过增加新的服务来实现演化。

SOA 服务质量的高低直接影响着 SOA 系统的整体质量，对分布在不同服务注册库中的服务进行有效的测定并给出相应服务质量的评价描述十分必要。本文旨在提出一种划分 SOA 服务质量等级的方法，即结合 SOA 服务的特点，给出质量评价的关键属性，并对服务质量进行等级划分，从而提高服务查找的效率，排除无效服务的干扰，为服务消费者提供高质量的查询结果。

1、SOA 服务特点

要评价 SOA 服务，需要首先明确 SOA 服务的特点，找出影响其质量属性的关键因素。

SOA 的关键价值之一就在于服务可以被复用的价值，也可以说复用是构成 SOA 的核心属性之一。通过复用获得降低开发维护成本，缩短应用交付周期，提升质量等种种好处。因此可复用性应作为反映 SOA 质量好坏的一个重要属性。而要保证其复用性，首先使得服务具备良好的封装性，即对于使用者而言，只需看到服务对象的功能列表。

松耦合：SOA 服务的关键还在于服务是松耦合和异构的，这一点区别与大多数其它组件框架。服务的使用者和提供者可以是分布部署的，可以位于不同的

系统平台上，可以使用不同的技术实现。应用程序开发者或系统集成者不需要知道服务的底层实现，只需要组合一个或多个服务即可以构建应用程序，例如，一个服务可以.NET 或 J2EE 实现，SOA 通过接口服务的标准化描述，隐藏了实现服务的细节，使得服务可以提供给任何异构平台和任何用户接口使用。

SOA 服务是独立的。软件的发展趋势是智能化，而智能化要求软件首先具有自己的独立性，意味着软件的生命周期要脱离其他软件的控制。无论调用者是否存在，服务本身是独立存在、独立发展的。这与一般的对象是不同的，一般的对象是由调用者创建并控制其生命周期的。每一个服务都能够提供相应的操作，能够很容易地被独立调用，其执行并不依赖于架构中的其他组件和服务。操作是通过标准方式封装和发布的，即只需要知道 service 如何满足我们的功能需求，而不需要管理它的生命周期，不需要理会那些维持 service 运行所需要考虑的种种细节。即对于 service 的了解只需要局限于功能接口即可，不要理会它的那些管理接口，配置接口等。

安全性：由于 SOA 服务放在 INTERNET 上并可以被任何人访问，而且为了方便企业间的协作，企业的业务也需要提供远程调用的能力，这就带来了许多潜在的风险。所以安全性必须作为一个整体性的要求提出。安全性的要求必须具备验证、授权、完整性、保密性和不可否认性等要素。

标准符合性：SOA 的实现强调基于统一的标准，SOA 系统建立在大量的开放标准和协议之上，以实现系统及信息的互联互通和互操作。因此，SOA 系统从规划到实施，标准都至关重要。

服务的互操作性：当前有多种创建应用程序的平台。但每种平台都习惯于使用自身的协议(本质上通常是二进制代码)来实现机器间的集成。因此，跨平台的应用程序在数据共享方面的能力相当有限。 “不论服务采用何种软件，使用何种硬件，都能够跨越这一传统的界限并无缝地集成在一起”这就是互操作性问题。

2、SOA 服务质量属性

在 GB/T 16260《软件工程　产品质量》中将软件质量属性划分为六个特性：功能性、可靠性、易用性、效率、维护性和可移植性，并进一步细分为若干子特性。在此基础上，针对 SOA 服务的特点，可以把 SOA 服务质量属性分为七类，

即在一般质量属性的基础上添加了“可复用性”，该属性又包括标准符合性、独立性、可复用性的依从性三个子特性，除此之外，SOA 质量属性重点强调了功能性中的互操作性和安全保密性的特点。具体内容可参见表 1：

表 1：SOA 服务质量属性

	功能性	可靠性	易用性	效率	可维护性	可移植性	可复用性
SOA 服务质量	适合性 准确性 互操作性 安全保密性 功能性的依从性	成熟性 容错性 易恢复性 可靠性的依从性	易理解性 易学性 易操作性 吸引性 易用性的依从性	时间特性 资源利用性 效率依从性	易分析性 易改变性 稳定性 易测试性 维护性的依从性	适应性 易安装性 共存性 易替换性 可移植性的依从性	标准符合性 独立性 可复用性的依从性

3、SOA 服务质量评定方法

假定 SOA 每一个服务质量属性的子特性对应的实际评价值为 p，并且该值只能在 0-1 之间取值（包括 0 和 1），根据其质量属性及其子特性的重要程度，分别赋予一定的权值，并用 W 和 w 来表示。设服务质量属性的个数为 m，每个质量属性中的子特性的个数设为 n，SOA 服务质量最终得到的评价值用 T 来表示。

则 $T=\sum_{j=1}^{m}\left(\sum_{i=1}^{n}p_i w_i\right)W_j$

根据对 SOA 服务质量中关键属性的分析，结合 GB/T 16260《软件工程　产品质量》的内容，对 SOA 服务质量属性及其相应的子特性赋予一定的权值（参见下表），对于其中的关键属性，将被赋予较高的权值。按照以上公式，算出 SOA 服务质量评价值。

表 2：　SOA 服务质量评价表

特性	特性权重（W）	子特性	子特性权重(w)	评价值	实际评价值(p)
功能	20%	适合性	20%	0-1	
		安全保密性	25%	0-1	
		准确性	20%	0-1	

		互操作性	25%	0-1	
		功能性的依从性	10%	0-1	
可靠性	15%	**成熟性**	30%	0-1	
		容错性	25%	0-1	
		易恢复性	25%	0-1	
		可靠性的依从性	20%	0-1	
易用性	5%	**易理解性**	30%	0-1	
		易学性	20%	0-1	
		易操作性	20%	0-1	
		吸引性	15%	0-1	
		易用性的依从性	15%	0-1	
效率	15%	**时间特性**	45	0-1	
		资源利用性	40%	0-1	
		效率的依从性	15%	0-1	
维护性	10%	易分析性	20%	0-1	
		易改变性	20%	0-1	
		稳定性	**25%**	**0-1**	
		易测试性	20%	0-1	
		维护性的依从性	15%	0-1	
可移植性	10%	适应性	25%	0-1	
		易安装性	15%	0-1	
		共存性	20%	0-1	
		易替换性	25%	0-1	
		可移植性的依从性	15%	0-1	
可复用性	25%	**标准符合性**	45%	0-1	
		独立性	45%	0-1	
		可复用的依从性	10%	0-1	

4、SOA 服务质量分级

在 SOA 服务质量评价的基础上，可以实现对 SOA 服务质量的水平进行等级划分。SOA 服务质量共分 5 个级别，其中以五级为最高级，一级为最低级。

SOA 服务质量等级划分参见表 3：

表 3：SOA 质量划分及要求

SOA 质量等级	对应实际评价值区间	总体描述
一级	X<0.3	差
二级	0.3≤x<0.5	较差
三级	0.5≤X＜0.7	中等
四级	0.7≤X＜0.9	良
五级	0.9≤X	优

5、小结

随着 IT 系统建设的快速变化和快速反应越来越成为企业成功的关健因素，SOA 架构也愈加引起了人们很大的关注，这种背景下，作为构成 SOA 系统的核心组件-服务，成为人们研究的焦点。因为其质量的好坏直接影响到整个 SOA 系统的质量，上文在 SOA 服务主要特点的基础上，分析了 SOA 服务质量的关键属性，结合 GB/T 16260，提出了 SOA 服务质量模型，并给出了 SOA 服务质量分级评价的方法。

小强系列之大话移动测试

作者：小强

测试界风云变幻，移动测试火爆来袭，移动测试真的像黄金价格一样吗？（黄金可是跌了啊）所有人都适合做移动测试吗？移动测试要怎么做？自动化是必经之路吗？好吧，面对如此多的疑问，小强带你浅入浅出看看移动测试。（纯属个人见解，无对无错，请各位看管理解）

1、概念

很多人都说手机测试，但手机测试到底要测什么，却没有几个人能完整准确的说出来。小强觉得，表达成 APP 的测试更为准确些。我们从以下几点来看：

- 首先，如果你就是用安卓原生你大可不必去测试安卓系统了，至少小强觉得没这个必要。
- 其次，如果像小米或锤子那样，做了二次的开发，那么有必要去测测，尤其是兼容性和稳定性，为什么这样说？就是因为小强是小米的用户，我想你懂得啊，伤不起。
- 再次，不论是什么安卓还是 ios，app 则是必测的东西，所以说我们常说的手机测试、移动测试本质上就是对 app 的测试。

2、手段

Ok，概念说完了，我想很多菜鸟都着急的问，要怎么测啊，要什么工具啊。其实，小强一直觉得测试不是一个技术活，也不是一个体力活，而是一个思维活。如果你没有良好的思维能力，测试你干不好。这里我们也从以下几点来看：

- 测什么、怎么测，需要针对具体的业务、产品、特点来分析，所以首先要深入理解你的产品才可进行下一步
- 当你理解了产品后，根据产品的特点来设计用例，其实这里大致可以分为三部分，一部分就是产品业务逻辑的测试，而一部分就是场景与平台的兼容性测试，最后则是先很多人搞噱头的性能测试（我们后面再说他）
- 好，那么针对上面的这几个方面，我们初步的测试手段就是手工+半自动化。有人说手工？你不要这么低级好不好。可是没有手工对业务、特点的了解，你如何去自动化？单纯的为了自动化而自动化，为了显示自动化多牛逼而自动化，小强看来没有任何意义。当你干一件事情的时候能从公司商业角度考虑，而不是只从技术角度考虑的时候你的 level 就会和别人不一样。
- 而对于半自动化而言，目前有不少好用、简单、实用的工具，如 AndTools，GT，各种云测平台，都是我们可以利用的，你要知道一个武林高手不是他武功有多牛逼，而是他能在各种环境中灵活的运用各种东西当成自己的武器。

3、自动化

接下来我们来谈谈这个，我只接触过 qtp 和 selenium，其他的并没有接触太多，可是据小强从多位童鞋那了解，自动化确实需要有编码的功底，不然真心的会很费劲，而且容易半路夭折。另外，小强一直觉得把一门技术学通了，其他的技术不是什么太大的问题，比如你把 selenium 学通了，你去学现在流行的 appium 难道就不会了？我相信，真正学懂的人 3 天就完全可以上手 appium。

所以自动化测试不是任何人都能学的，入门是有难度的。另外，对于 monkeyXXX 而言有的人就觉得简单的无比，可有人就觉得难，这个我们没法去衡量，但是小强想说的是，我们学什么都要为了实际工作中效率、投入产出比最大化的方向走，记住，最好的不一定合适，合适的才是最好的。

4、性能

总有人问手机性能怎么测？其实小强自己也有点疑惑。首先，app 的性能小强觉得可以从两个纬度来看，一个就是重复频繁操作的时候性能表现如何；另一个就是前端大量请求，后端服务的性能表现如何。

那么对于第一个纬度而言，完全可以利用脚本模拟完成，利用辅助软件记录相关信息，最后分析。而第二个纬度，完全就是和我们的 WEB 性能测试一样的……这里大家要注意了，第二个纬度是关注后端的性能，道理和 WEB 性能测试一样。LoadRunner 也完全可以胜任。

所以，我想说的是，不论怎么样，只有认识到本质才能快速的切入，不然你永远都像一个无头苍蝇到处乱撞，任人宰割。

到此为止你是不是可以大概的、浅浅的理解了？如果你可以，我很安慰，那么你就可以按这种思路来试试，也许你会体会更深。

结尾处，再次说明，只是个人见解，不要太过纠结，提供思路而已，任何事情都有一定的发展规律与步骤，一口吃成大胖子不可行，但愿移动测试能健康长久的发展下去，祝福好运！

【测试人生】两年半的成长之路

作者：李飞波

2010 年的今天，对软件测试这一行业及岗位还懵懵懂懂的我，把目标定位在会计界。而由于非会计专业，毕业前夕的求职，累累碰壁。2011 年初，某大学一次招聘会的尝试，改变了我的职业方向。

2011 年 5 月份有幸同时接到了两份测试相关工作的 offer，公司 A 是一家比较大型的房产公司，产品有 OA 系统、公司内部网站等，公司 B 一家中型电子商务公司，产品主要是外贸电子商务网站，当时由于公司 A 需从实习开始，并且待遇薪水是公司 B 的一半，而公司 B 让从试用期开始。不得不承认，当时考虑问题的欠缺性，看重眼前利益，最终选择了公司 B。

【建议】:

作为刚毕业的应届生，在求职的过程中，不要太看重眼前公司可以给我们多少报酬。作为一名刚步入社会的职场新人，主要看该公司的发展前景如何、在里面可以学到多少东西，好好沉淀一段时间，提高自己的测试技能，无论的新东家或旧东家，总会看到你的闪亮点，到时涨薪水固然是自然而然的事情了。

入职公司 B 后，主要从事 WEB 网站的黑盒测试，发现所接触到的东西远比想象中少。每天一上班，就用 FF 的 Launch Clipboard 插件，alt+shift+K 快捷键批量检查那几百个站群是否可正常访问，页面图片是否加载正常；每天总有那么一些报 502，统计出来反馈给开发经理；一个多小时的时间这么机械的过去了；工作相对轻松，没什么压力可言。但当时的心态很浮躁，总感觉每天就鼠标点点、所做的工作没技术含量、在公司没什么可以学习。经常在网上悄悄投几份简历，适当时机找个借口请假出去面试，抱着一颗不会在这久呆得过且过的心态，就这么过了大半年。而 2012 年初，部门突然提出使用 Selenium 进行自动化测试，基于想学习这个，决定继续留下来一段时间再裸辞。

【建议】:

作为一名新的测试员，一开始接触到基础的东西，不要认为就没有了技术含量，从而产生浮躁的心态，想着一定要接触什么功能自动化、性能测试才是最好的，其实不是这样，黑盒测试，如果想深入，做到最好，也有很多东西需要我们掌握。

我们不能改变环境，但我们可以适应环境；如果新入职的公司，不如所想，就需要及时地调整自己的心态及方向。

2012 年中下旬，终于换了第二东家。在新东家，测试人员需要搭建测试环境及维护。而大学时代尚未接触过 linux 系统的我，当时有点担心胜任不了。带

着既兴奋又紧张忐忑的心情，接受了这份 offer。在新的公司，有独立的测试部门，测试规范、测试流程、测试管理都较为严格，leaders 都是完美主义者。

关于测试流程：从需求提议、评估测试周期、需求熟悉、前期准备、执行测试、测试报告等。

需求提议：对于面试时，可能大家都有被问及的问题，测试是何时介入？我们的答案可以说是产品提需求时。针对产品需求频频变动及较好理解产品的问题，一般我们 testers 都会参与中大型项目的需求讨论会议。除了作为开发与产品讨论需求的听众外，我们也要从测试的角度，提出疑问，或帮助完善产品的设计。

评估测试周期：主要从三方面进行评估，一、测试内容的多少；二、测试内容的复杂度；三、测试人员的安排。

需求熟悉：对于测试周期较紧急的项目，目前一般不要求写测试用例，而本人也比较纠结写测试用例，总感觉对于真正的测试执行没起到很大的帮助。但 checklist 还是有一定的帮助的。拿到需求时，时间允许的情况下列出 checklist，明确本次测试需特别关注的点。

前期准备：测试环境的搭建或更新、测试计划安排等等。

执行测试：虽然与上一家公司都是 WEB 网站测试，但涉及到的东西较多，应用的关联性也较强。作为测试员，不仅仅是测试功能实现的正确性，还包括数据显示的正确性、接口测试、定时任务测试等。目前也初步接触了 LR 性能测试、安全测试等，但技术之路还漫漫长兮。

【总结】

1、测试不仅是发现问题的过程，还需尽量找到产生问题的原因；

2、测试时，先整体功能流程再细节问题；

3、多思考如何提高测试覆盖度及测试效率；

4、作为项目负责人需把控好项目的进度及合理安排测试时间及任务；

5、对于需求不明确的地方，多与用户或开发进行沟通确认；对于一些优化性问题，提出问题的同时，带上建议的解决方案；

6、不能尽信开发人员，要有自己的见解。

更多精彩内容请见　原创测试文章系列（三十二）（下篇）