

# AC 应用案例：金融系统自动化测试

## 目 录

AC 应用案例：金融系统自动化测试 .....	1
1. AC 应用案例 1：金融系统自动化测试 .....	1
1.1 金融系统业务测试的复杂性.....	1
1.2 使用 AC 的观点完成 TestJob 的定义.....	2
1.3 AC 提供 QTP 的工厂开发模式 .....	4

作者：柳胜

## 1. AC 应用案例 1：金融系统自动化测试

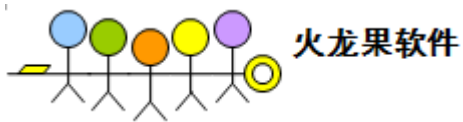
对于金融业务系统来说，测试案例往往涉及到数据校验，交易确认，业务关联等等，手工测试执行起来比较复杂，更不用提自动化测试的实施了。这也是当前金融系统业界功能自动化测试程度不高的原因之一。

### 1.1 金融系统业务测试的复杂性

比如某个银行支付系统的转账案交易，手工测试大概的流程如下：

1. 创建 Test Account A 和 Account B，并在各自名下建立相应权限的转账卡
2. 使用 Account A 登录银行转帐系统，使用名下某一张卡对 Account B 做转账交易，支付金额为 1000 元人民币。
3. 查看 Account A 和 B 的余额，确认 A 账户减少 1000 元，B 账户上增加 1000 元。

以上三个步骤从银行业务角度来看是各自独立的三个功能，但在转账场景里，又有密切的联系。步骤 2 依



依赖于步骤 1 的先决运行，步骤 3 则需要步骤 2 的转账数据。

使用 QTP 等工具针对以上案例开发脚本，则会面临棘手的问题，如果把三个功能写在一个脚本里，就会大大降低每个功能脚本的复用性。若开发成三个不同的脚本，那么彼此的关系和数据交互又需要增加额外的开发成本来实现。

### 1.2 使用 AC 的观点完成 TestJob 的定义

在 AC 的世界里，一切都得非常简单，三个功能将被定义成三个 TestJob。

Create\_Account\_Info 负责创建测试账户 A 和 B，然后将 accountA，accountB 作为参数输出。在 AC 中做如下定义：

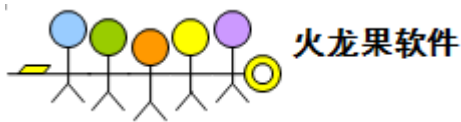
```
<TestJob name="Create_Account_Info" description="以管理员身份登录后台系统创建测试账户及相关卡信息 " depends=""  
driver_type="QTP">  
  
    <JobOutput name="accountA"/>  
  
    <JobOutput name="accountB"/>  
  
    <Lib location="common\lib\lib_utility.vbs"/>  
  
    <Run path="testcase\qtp\admin_createaccount"></Run>  
  
</TestJob>
```

其中 admin\_createaccount 是录制好的 qtp 脚本路径，lib\_utility.vbs 是脚本使用到的 lib 文件，AC 将会自动加载到 QTP 运行环境中。

Transfer\_FromAToB 则会运行转账交易，将 account A 里的款项转给 account B，并将转账数额作为参数输出。在 AC 中做如下定义：

```
<TestJob name="Transfer_FromAToB" description="以账户 A 登录，转账给 B 账户" depends=""Create_Account_Info"  
driver_type="QTP">  
  
    <JobInput name="accountA"/>  
  
    <JobInput name="accountB"/>  
  
    <JobOutput name="transfer_amount"/>  
  
    <Lib location="common\lib\lib_utilityvbs"/>  
  
    <Run path="testcase\qtp\transfer_bank"></Run>  
  
</TestJob>
```

Verify\_Account 根据输入的两个 account 信息和转账金额，检查 account 的余额是否预期变化。在 AC 中做如下定义：



```
<TestJob name="Verify_Account " description="检查账户 A 和账户 B 的余额是否预期变化 " depends="" Transfer_FromAToB  
" driver_type="QTP">  
  
    <JobInput name="transfer_amount"/>  
  
    <JobInput name="accountA"/>  
  
    <JobInput name="accountB"/>  
  
    <Lib location="common\lib\lib_utilityvbs"/>  
  
    <Run path="testcase\qtp\transfer_bank"></Run>  
  
</TestJob>
```

以上三个 TestJob 被 AC 组织起来，将会根据 depends 关系计算出执行路径：

Create\_Account\_Info → Transfer\_FromAToB → Verify\_Account

同时，AC 搭建一条全局数据通道，所有 TestJob 的 JobInput 和 JobOutput 等数据都可在这条通道中进行交互。针对 QTP 脚本，AC 提供了框架 vbs 函数 writeInitoACChanel(paraname,paravalue) 和 getDataFromChannel(paraname)实现写入和读出全局数据的功能。

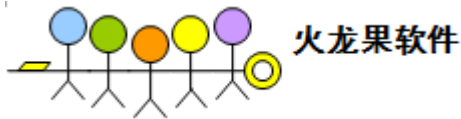
上面谈到的是一种理想的 TestJob 结构模型，在实际的业务中，还有一些比较复杂的因素。比如，Create\_Account\_Info 创建测试账户，测试卡号等信息这些工作，更合适在产品实例安装结束之后，作为基础数据被 sql 脚本直接创建至 Database 中。

这时，我们可将 Create\_Account\_Info 从 QTP 改由 Java 类型，而接口维持不变。

```
<TestJob name="Create_Account_Info" description="Java 程序调用 jdbc 运行 sql 脚本,在数据库中创建基础数据" depends=""  
driver_type="JAVA">  
  
    <JobOutput name="accountA"/>  
  
    <JobOutput name="accountB"/>  
  
    <ClassPath location="javacase \jdbc_sql.jar"/>  
  
    <Run path="jdbc.createAccount"></Run>  
  
</TestJob>
```

其中 jdbc.createAccount 是按照 AC 规范自开发的 java 程序，调用 jdbc 运行 sql 脚本。

这种好处显而易见，Create\_Account\_Info 从 QTP 转换成 Java 类型之后，只要接口维持不变 (JobOutput 不变)，那么其它两个 TestJob，Transfer\_FromAToB 和 Verify\_Account 不会受到任何影响。



同理，Verify\_Account 也可改写成 Java 或者 selenium 类型，以不同的方式进行验证。

另外一种复杂的情形可能会出现在 QTP 层面上，如果 QTP 的脚本规模较大，而又在同一个产品实例的上下文中完成不同的功能。比如，登录后做查账，查账之后做理财交易等等，都是基于一个 web session 上完成的。针对这种情况，AC 的 QTP 提供了一种 Factory Mode（工厂模式）的开发方式，使得所有的测试案例的定义和执行可以在同一个 QTP 执行环境中完成，非常适合 QTP 大规模的脚本开发工作。

### 1.3 AC 提供 QTP 的工厂开发模式

基于录制生成的 QTP 脚本，是面向功能的，而不是结构化的测试案例。这使得 QTP 在维护和增加测试案例时，成本十分昂贵。为此，AC 引入工厂开发模式，使得 QTP 的开发像 Junit 一样清晰方便。

QTP 的工厂开发模式有如下规范：

1. 每个 QTP 的测试案例在表现形式上都是一个 Vbscript 的函数，测试案例的增加/删除通过增加/删除一个 VBS 的 Function 来达到。
2. 工厂模式不支持对象库模式的脚本，所有的功能都以 Description 编程来实现
3. 使用 checkDependence 函数来检查每个测试案例的运行结果状态
4. 调用 writeIntoACChannel 和 getDataFromChannel 来完成测试案例之间的数据交互。
5. 每个测试案例都是一个函数，一个函数是否成为一个测试案例取决于在 TestJobFile.xml 中的定义。

示例：

QTP 自带的 Flight 演示程序，录制生成的脚本如下模式：

‘登录客户端

```
Dialog("Login").WinEdit("Agent Name:").Set "testing"
```

```
Dialog("Login").WinEdit("Password:").Set "mercury"
```

```
Dialog("Login").WinButton("OK").Click
```

‘输入机票信息，下订单

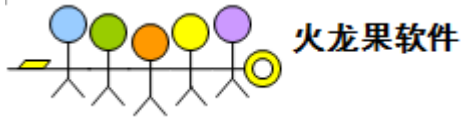
```
Window("Flight Reservation").ActiveX("MaskEdBox").Type "081210"
```

```
Window("Flight Reservation").WinComboBox("Fly From:").Select "Frankfurt"
```

```
Window("Flight Reservation").WinComboBox("Fly To:").Select "London"
```

```
Window("Flight Reservation").WinButton("Insert Order").Click
```

```
Window("Flight Reservation").Close
```



从 QTP 录制脚本转换成工厂模式脚本，步骤如下：

1. 创建一个 testcase.vbs
2. 将原始脚本进行 description 改写，并按照工厂模式规范，写入 testcase.vbs

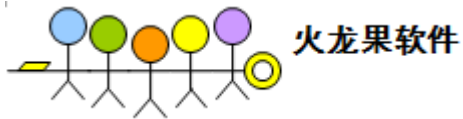
### **Function login**

```
'调用框架函数 ReportRunningInfo，写日志
ReportRunningInfo "start to run login test case"
.....Description 创建.....
Dialog(dlgLoginDesc).WinEdit(editUserDesc).set getDataFromACChannel("user")
Dialog(dlgLoginDesc).WinEdit(editPasswdDesc).set getDataFromACChannel("passwd")
Dialog(dlgLoginDesc).WinButton(btnOKDesc).Click
if(Window(flightWindowDesc).Exist(30)) Then
    '调用 reportPass，向 AC 报告当前案例成功状态
    reportPass "has signin in successfully"
    login = True
Else
    '调用 reportFail，向 AC 报告当前案例失败状态
    reportFail "failed to signin in"
    login = False
End If
'将用户名写入数据通道，供后续执行的测试案例使用.
username = "sheng.liu"
writeIntoACChannel "displayname",username
```

### **End Function**

### **Function bookFlight**

```
ReportRunningInfo "start to run book flight test case"
'检查 login 案例是否成功，如果失败，当前案例则返回失败.
If Not CheckDependence("login") Then
    bookFlight = False
Exit Function
```



*End If*

.....*Description* 创建及调用.....

'从数据通道中获得 login 案例写入的用户名，作为订单用户名下单。

`username = getDataFromACChannel("displayname")`

`Window(flightWindowDesc).WinEdit(nameEditDesc).Set username`

`Window(flightWindowDesc).WinButton(insertBtnDesc).Click`

.....

### **End Function**

3. 定义 TestJobFile.xml 文件，指定工厂模式和测试案例。

```
<TestJob      name="QTP_DesktopClientTest_AC"      description="demo"      factoryMode="true"      depends="Java_Init"
driver_type="QTP" iteration="">

  <Lib location="testscripts\DesktopClient\testcase.vbs"/>

  <Testdata type="xsl" location="data\data_global_shining.xls"/>

  <Testdata type="xsl" location="data\data_global_shining.xls"/>

  <Testdata type="iteration" location="data\testdata.xls"/>

  <TestJob name="test.vbs" description="demo" depends="">

    <TestJob name="login" description="login flight app"><Run path="login"/></TestJob>

    <TestJob name="bookFlight" description="book flight"><Run path="login"/></TestJob>

  </TestJob>

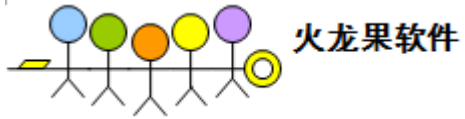
</TestJob>
```

4. 运行 AC 框架

AC 会从指定的 lib 路径中获得 testcase.vbs，然后以工厂模式运行 login 和 bookflight 两个测试案例。并最终形成测试报告。

从上面的示例可以看出，经过工厂模式改写后，QTP 自动化测试具有以下优势

1. 测试案例的粗细粒度更加细微，可以有效地与手工测试案例形成一一对应的映射关系
2. 扩展性大大增强，增加新的功能点，只需开发新的 function 函数即可集成到 AC 中。
3. 案例之间的依赖关系和数据交互更加密切。



总之，工厂模式非常实用于大规模的 QTP 的自动化测试脚本开发，大大减少维护成本，提升开发效率。

有关自动化测试框架的更多内容，请参看本文前传《[自动化测试基础理论及实施经验](#)》

### 参考文献

《软件自动化测试框架设计与实践》 柳胜著 人民邮电出版社

**作者：**柳胜：计算机应用硕士，有近十年的软件测试经验。曾在摩托罗拉，oracle 等企业担任高级开发工程师，高级测试工程师等职位。

### 著书：

《性能测试从零开始-loadrunner 入门》 2007 年 电子工业出版社

《软件自动化测试框架设计与实践》 2009 年 人民邮电出版社