目 录

ERP 软件测试中条形码测试

作者：薛继国

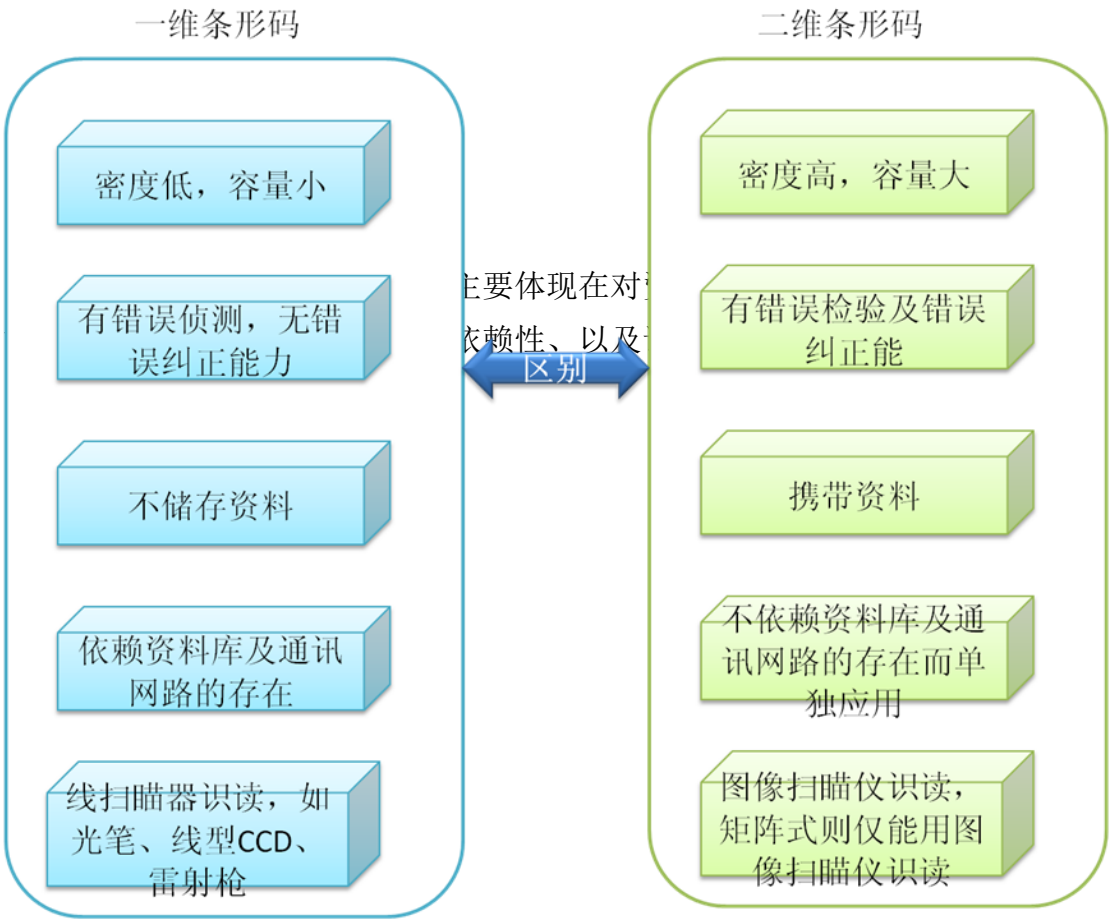
摘要：条形码技术经过多年的发展，已经在各行业商品中广泛应用。ERP 领域竞争激烈，为了使实物操作和 ERP 数据录入形成有机的结合而引入条码操作。仓库中引入条码，可以大大的提升数据的准确性和及时性。以及降低对仓库搬运工人对应用计算机能力的要求和工作强度。通过扫描物料上的条码，直接生成仓库的出入操作单据和盘点单据，同时对于有批次管理，序列号管理，货位管理等企业，同样可以支持扫描这些信息，提高工作效率。基于条形码应用特点，需要有针对性测试，制定详细的测试计划策略，编写完善的用例才能保证产品的正确性。

关键字：条形码；拆单；拆记录；成对测试

一、条形码

条形码（barcode）是将宽度不等的多个黑条和空白，按照一定的编码规则排列，用以表达一组信息的图形标识符。常见的条形码是由反射率相差很大的黑条（简称条）和白条（简称空）排成的平行线图案。条形码可以标出物品的生产国、制造厂家、商品名称、生产日期、图书分类号、邮件起止地点、类别、日期等许多信息，并且条形码的使用非常经济实用，能够提高日常库管员的录入速度，可靠性和安全性非常高，采集的信息量非常大，同时具有纠错功能，因而在商品流通、图书管理、邮政管理、银行系统等许多领域都得到了广泛的应用。

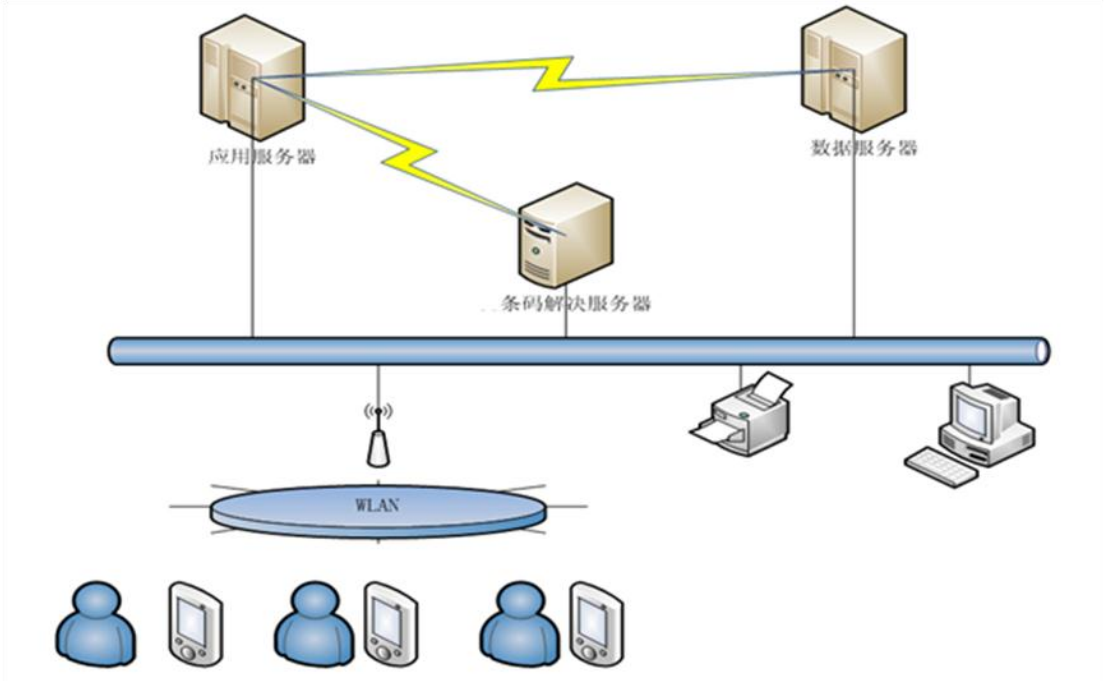
条形码分为一维、二维条形码，虽然一维和二维条码的原理都是用符号（Symbology）来携带资料，达成资料的自动辨识。但是从应用的观点来看，一维条码偏重於“标识”商品，而二维条码则偏重於“描述”商品。因此相较於一维条码，二维条码（2D）不仅只存关键值，并可将商品的基本资料编入二维条码中，达到资料库随着产品走的效益，进一步提供许多一维条码无法达成的应用。例如一维条码必须搭配电脑资料库才能读取产品的详细资讯，若为新产品则必须再重新登录，对产品特性为多样少量的行业构成应用上的困扰。



E 国际物 于世界各 条形码 使用的也 EAN— 3（缩短版） 前缀部 品代码和 来标识 码权在国际 拿大。)-692代表中国大陆，471 代表我国台湾地区，489 代表香 码表

一维条形码

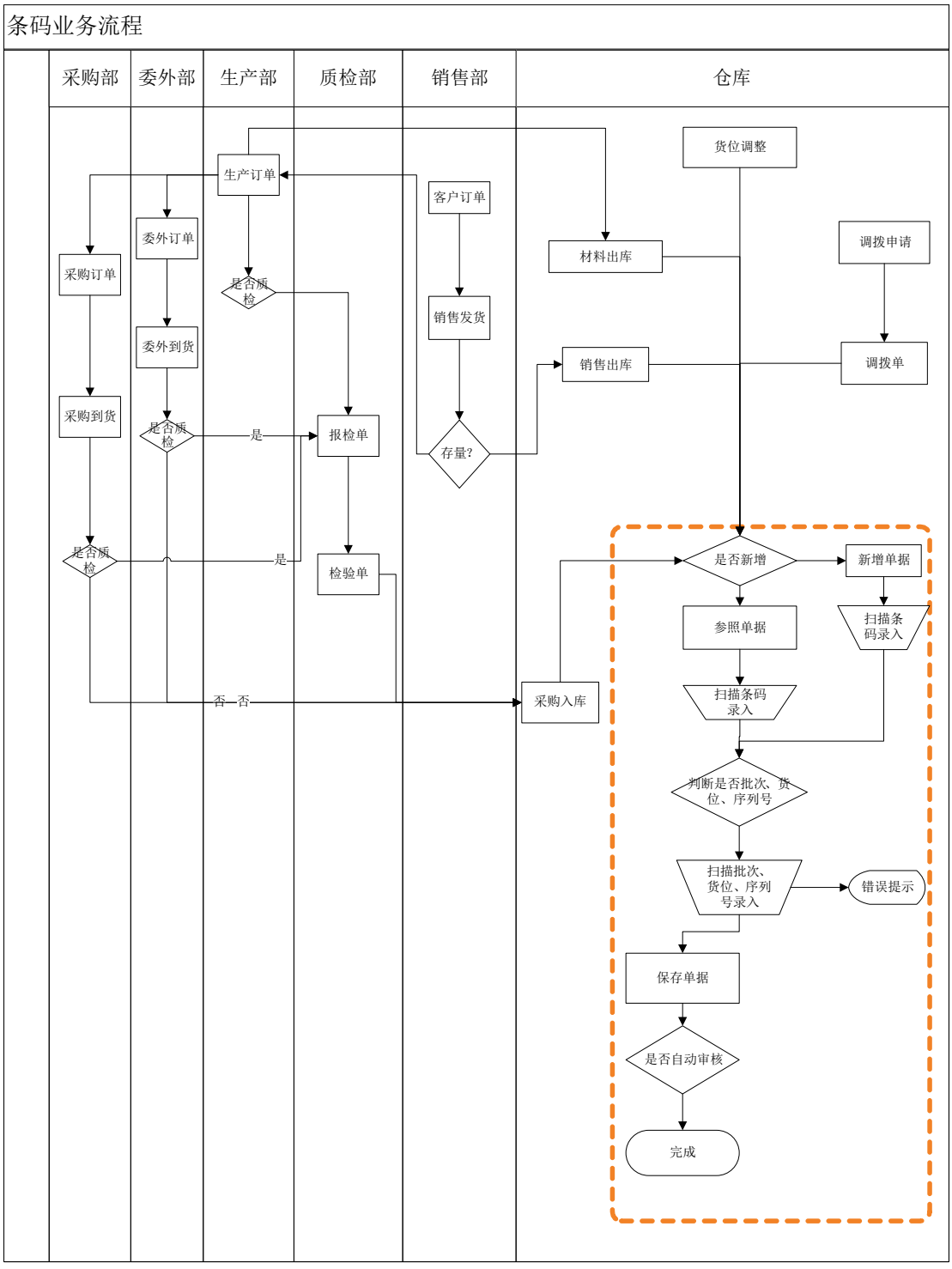

二维条形码



本文以一维条码在 ERP 系统中应用为例，对于商业企业或工业企业来说，仓库往往与企业在不同的区域，有的企业仓库在不同的城市，ERP 软件服务器通常部署到企业总部，在仓库部署 ERP 的客户端，当商品需要出入库、盘点、货位调整时，库管员通常依据单据在 ERP 系统中进行录入维护，且通常库管员的素质较低，此工作效率较低，而且容易出错。通过引入条形码的应用，条形码系统在仓库内进行部署，通过无线方式与总部服务器进行连接，并进行数据交换。库管员对于质检入库的商品，通过终端扫描器进行实物商品条形码扫描，自动识别商品信息带入到终端程序，通过无线交互传递到服务器上，形成对应的单据，同时传递过程中进行数据校验，保证传递信息的正确性

ERP 软件中条形码应用方式主要有三种方式：固定扫描终端（终端无程序）、无固定扫描终端（终端无程序）、有固定扫描终端（终端有程序），三者区别主要是软件是否限制固定硬件，硬件终端是否有程序需要维护录入数据，下文重点以第三种方式为例进行测试阐述。

三、条形码业务流程

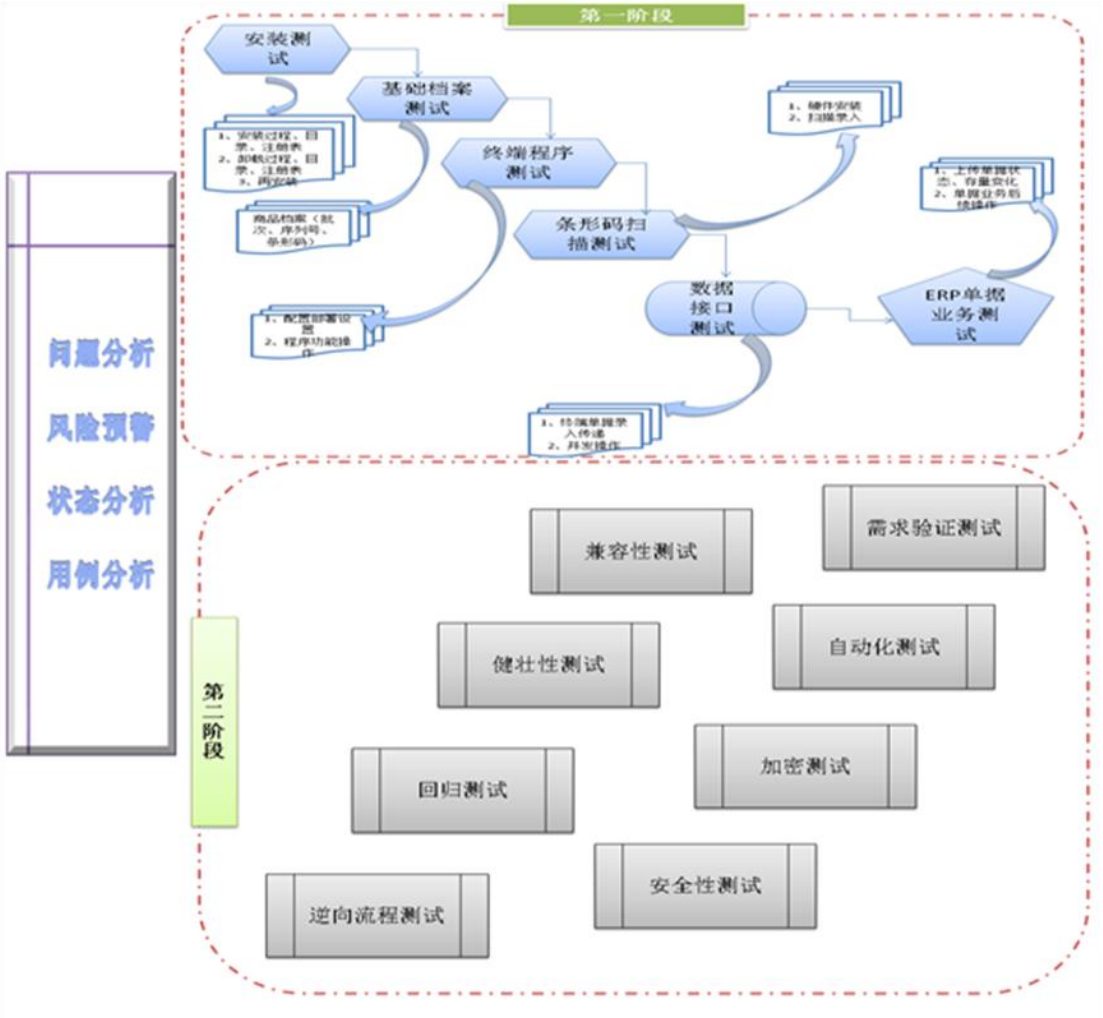

条码业务流程
采购部
委外部
生产部
质检部
销售部
仓库
生产订单
客户订单
货位调整
委外订单
材料出库
调拨申请
采购订单
是否质检
销售发货
销售出库
调拨单
委外到货
采购到货
是否质检
是
报检单
存量?
是否质检
是
检验单
否
否
采购入库
是否新增
新增单据
参照单据
扫描条码录入
扫描条码录入
判断是否批次、货位、序列号
扫描批次、货位、序列号录入
错误提示
保存单据
是否自动审核
完成

- 销售部业务员与客户签订销售合同，并在系统中维护销售订单，销售主管审批确认，并生成相应的销售发货单及审批
- 仓库库管员依据已审批的销售发货单进行先存量核对，如果存量足够则通过扫描器进行实物出库并同时生成销售出库单传递到服务器
- 如果仓库存量不够，需要通知生产部进行生产，生产部接到生产通知，在系统中依据销售订单下达生产，形成生产订单；依据生产订单在仓库进行材料零用生产，库管员依据生产订单进行扫描器扫描实物出库，同时生成材料出库单传递到服务器
- 如果仓库需零用材料不足，需通知采购部门进行采购，采购部接到采购通知，在系统中依据销售订单开采购订单进行采购；采购到货后进行质检入库，通知生产部进行材料零用
- 对于销售商品可以委外其它厂商进行生产，因此可以在系统中维护委外订单，依据委外订单在仓库中进行材料零用，同时生成对应材料出库单传递到服务器；生产完毕后质检入库，仓库管理员对实物进行扫描录入并传递到服务器生成产成品入库单
- 企业如果有货位管理，且需要在不同货位间进行实物移动，库管员进行实物扫描录入货位调整单并传递到服务器
- 企业月末进行盘点，在 ERP 系统中维护盘点表，库管员通过扫描终端参照盘点表进行实物扫描盘点并传递到服务器上，更新的对应的盘点表的实盘数量

四、条形码测试框架

一维条形码主要是标识商品，因此在软件中需要维护商品与条形码对应关系，以便软件能够通过条形码来识别商品，在商品录入时能够自动带入相关信息。企业中商品应用比较复杂，通常商品具有批次要求，且在库存管理方面有货位要求，因此基础资料准备是保证测试的全面性和业务的吻合性。

第三种应用方式中条形码需要有扫描终端进行识别录入，终端有相应的程序进行单据维护录入，为了保证录入的数据能够传递到后台软件数据库中，需要必要的接口程序进行数据链接，为了保证传递数据的正确性，需要有针对性进行安装和终端程序及传递数据的测试，基于以上情况，整体测试框架为：



五、条形码测试数据设计

条形码应用相对复杂的是终端扫描后维护终端程序中的单据，然后传递到后台 ERP 系统中，在此过程中较容易出现错误。企业的库存管理在实际应用时录入的商品往往是不固定的，有时依据入库和出库的商品不同，每种商品有不同的货位和批次，一张单据中经常会存在多个存货、多个批次的情况，而且部分出入库是参照上游订单或发货单生成，因此存在上下游单据间的一对一、一对多、多对

一、多对多的情况，因此测试用例设计重点是关注不同业务场景的数据，可以从以下几方面考虑测试数据设计：

1、商品属性组合

不同商品属性在实际业务中控制是不同的，而且用户实际商品往往具有不同的属性，因此为了保证各种不同属性都能满足业务控制要求，需要测试每个属性单独使用正确性，然后再测试不同组合的正确性，但多个属性组合情况比较多，既要提高测试覆盖度，又要提高测试效率，减少测试工作量，需要使用成对测试法来设计用例数据，商品属性包括：批次、序列号、货位、质检，将这些属性作为成对测试法的因素，每种因素各有两种因素值，所有因素值组合共有 16 中情况，但使用成对测试法计算得出的用例只有 6 种情况即可以满足测试要求

有批次，有货位，有序列号，质检，
有批次，无货位，无序列号，非质检，
无批次，有货位，序列号，非质检，
无批次，无货位，无序列号，质检，
有批次，有货位，无序列号，质检，
有批次，无货位，有序列号，质检，

以上是针对必须有条形码的情况，对于无条形码的存货，如果扫描的条形码在商品档案中没有对应有相应提示控制，因此只需要准备一个无条形码的存货，可以没有以上属性即可。

2、不同换算率

换算率共有三种：无换算率、固定换算率、浮动换算率。固定换算率是指有多个辅计量单位，每种辅计量单位的换算率是固定的，在实际业务单据操作中是不能修改的，有些商品可能有不同的包装方式，每种包装方式的换算率是不同的；浮动换算率是指商品只有一个辅计量单位，但该单位的换算率在业务单据操作中是允许修改。主辅计量之间的换算关系：辅计量单位=主计量单位 / 换算率 ，因此准备基础商品时三种换算率的商品都需要准备

3、上下游单据关系

入库单参照采购订单或生产订单入库，出库单参照销售发货单出库，在此参照中既可以拆单，也可以拆记录生单，因此上下游单据间存在一对一、一对多、多对一、多对多的情况，因此准备多张上游单据，即：订单 DD001 生成入库单 RD001，订单 DD002 生成入库单 RD002、RD003，订单 DD003、DD004 生成入库单 RD004，订单 DD005、DD006 生成入库单 RD005、RD006

4、商品存量

商品出入库时会对存量进行更新，出库时减少库存现有存量，入库时增加库

存现有存量，这个更新时点是在终端单据上传到 ERP 系统中时进行，如果存量不够会造成单据无法生成，会有正确的判断提示，因此在准备存量是需要从三个方面准备：无存量、有存量不够出库、有存量够出库。

之所以需要考虑无存量，因为对于商品没有任何出库入库记录，即初始化状态，数据库表中的存量字段为 null，开发代码需要将 null 转化为可计算值，即：isnull（currentstock，0），这个转化是经常出错的地方，出错主要在于开发漏转而造成产品异常。

有存量不够出库，在产品单据保存时程序会判断是否够出库、是否允许出库，会给出正确提示和控制，如果存量不够不允许出库，则单据保存时会控制保存不成功并有正确提示，如果存量不够且允许超量出库，则单据保存时可以保存成功，但会给出正确提示，因此为了测试这几种情况需要准备商品有存量但不够出库的数据

有存量够出库，在单据保存时会更新存量，存量会增加，即：currentstock_B=currentstock_A+iquantity，因此需要测试存量代码计算逻辑的正确性，存量增加的量是否与单据的数量一致

5、BOM 结构

对于加工型企业，商品下订单时所需子件通常依据 BOM 结构来生成的，不同的子件使用量、以及使用类型往往是不相同的，在生产订单材料领用时所生成的材料出库单会带入订单的子件数量，而且对于多子件的生产订单可拆单、拆记录进行领料，多次领料会减少订单领料数量，为了测试数量的计算逻辑正确性，需要准备多子件的 BOM 结构

6、业务是否完成

扫描终端单据可以参照上游单据进行生单，但前提条件是上游单据未被完全参照生单，即业务未完成，如果已经完成的上游单据是否不允许被再次参照的，因此需要在后台 ERP 系统中提前准备未生单、已生单未完成、已生单已完成的上游单据，这其中很重要的一点是每张单据需多记录，这些多记录中有未生单、已生单为完成、已生单已完成的情况，这样才能满足拆单、拆记录、多对多的关系测试。

7、业务规则控制

部分企业库存管理比较严格，在实际业务出入库时有严格控制，不允许超可用量出库、不允许超单出入库，因此 ERP 系统有相关选项来控制，扫描终端在增加单据参照来源单据时需要依据选项控制规则，同时在上传单据到 ERP 时也要同步进行判断，这也涉及到选项随时改变后对业务的影响，准备数据要重点考虑，覆盖各种选项组合的情况。

六、条形码测试要点

ERP 软件产品最终是要满足客户的使用，而且作为水平产品需要考虑各种不同企业用户使用场景，因此决定了产品测试方向，不能盲目无目的的测试，需要依据产品功能特点以及用户可能出现的应用场景来分析进行测试，对于条形码的应用来看，最重要的是终端程序的部署和数据上传、以及终端扫描程序的应用，因此可从以下几个方面考虑测试点：

1、安装卸载测试

扫描终端可以维护单据并上传数据到 ERP 系统中，因此决定了必须存在必要的程序来满足之间的接口，需要在 ERP 系统端服务上安装相关接口程序，通常一个 exe 或是 msi 安装测试无非分为三种情况：安装测试、安装后卸载测试、卸载后再安装测试，同时在此过程中需要进行必要的病毒扫描测试，避免有病毒程序带给客户

安装测试包括：安装前、安装中、安装后，安装前需要测试安装文件，包括安装文件大小、名称、病毒扫描文件等；安装中主要测试安装过程中交互界面显示及操作正确性、以及安装过程中病毒防火墙杀毒；安装后需要测试安装后文件目录位置及名称正确性、文件目录结构正确性、以及目录文件病毒扫描测试

安装后卸载测试：测试卸载方式，包括操作系统卸除、程序菜单调用卸除、重新运行 setup 卸除，三者应都能进行卸载且成功，重新运行 setup 卸载时过程交互界面选择及显示正确性；卸载后相关目录文件是否卸载正确，应只保留规定文件，如：新产生的 logs 文件等；

卸载后再安装测试：测试再安装时文件注册正确性；测试未删除原有安装目录和数据库时目录覆盖和数据库覆盖正确性；安装产品功能测试，有时安装过程正确，但个别文件没有覆盖成功或注册成功，会造成产品功能不能使用或报错，因此需要测试再安装后产品流程功能正确性

2、环境兼容性测试

在 ERP 端需要安装相关程序，且终端有相关程序，这些程序能否在各操作系统中进行安装卸载，以及安装后产品使用需要进行测试

3、异常健壮性测试

企业在实际业务应用中是在仓库部署设置，使用终端在各货位间操作，终端与服务器是无线链接，因此存在断网情况，在数据上传过程中如果断网应该进行相关处理，针对此项需要进行必要测试；另外，上传后台服务器时如果终端断电或服务器断电时也同样需要进行处理，机制回滚操作，保证数据正确性，无垃圾数据产生，不能影响原有数据继续使用，所以针对这两种情况需要时时上传时人为模拟，即手工破坏网络和电路，造成断网、断电现象，以此来测试。

4、终端非正确操作测试

非正常操作情况下程序应该给出正确提示，结果可能情况有两种，一是提示确定后返回初始状态，二是提示确定后返回编辑状态，无论是哪种状态，都不能有非常提示，都需要代码中进行错误处理。一般情况下可能产生的错误也有三类：

① 产生错误提示，提示确定后不影响使用；

② 产生错误提示，提示确定后产品自动退出；

③ 产生错误提示，提示确定后程序宕掉；

④ 产生错误提示，提示确定后产品未自动退出，但其它相关功能不能使用。

其中第四种错误情况主要程序代码没有回滚，数据库表被锁死，产生死锁，造成其它功能无法访问，影响功能使用。

5、逆向流程测试

逆向流程是针对正向流程来说，正向是指实际业务的正向，如：增加单据-》审核-》下游参照生单-》结算-》核销等业务流程操作。对于条形码的逆向流程测试主要是扫描终端参照单据生单传到后台，后台 ERP 系统中删除所生成单据，在扫描终端再次参照生单，测试其删除后是否可以再次参照，带入的单据信息是否正确，相关量的信息是否已经因为删除单据而减少。

6、加密测试

条形码加密集成在 ERP 产品加密中进行统一控制，无加密狗时同样判断是否超演示期，判断依据是单据日期超过 2 个月过期无法使用，与 SCM 产品判断依据基本一致。若有加密狗依据加密狗许可数判断站点登录情况，超过许可数不允许登录。因此条形码加密测试主要体现在：

无加密狗，无单据超期：应可登录，登录后是演示版

无加密狗，有单据超期：不可登录，以及提示正确

有加密狗，无单据超期， 登录站点数<=许可数：可登录，登录非演示版，加密服务中显示加密狗信息正确，以及登录站点数正确 1/N（“1”表示登录站点数，“N”表示许可数），站点退出程序后显示正确 0/N

有加密狗，有单据超期， 登录站点数<=许可数：可登录，登录非演示版，加密服务中显示加密狗信息正确，以及登录站点数正确 1/N（“1”表示登录站点数，“N”表示许可数），站点退出程序后显示正确 0/N

有加密狗，有单据超期，登录站点数>许可数：未超过许可数的站点可登录，超过许可数站点不可登录，加密服务中显示加密狗信息正确，以及登录站点数正确 N/N（“N”表示许可数），站点退出程序后显示正确 N-1/N

7、数据接口测试

数据接口是指后台与扫描终端数据的传递，即：扫描终端从后台取数后显示

的数据，后台显示从终端传递的数据，传递的数据对存量的影响，以及删除后对存量变化和相关单据信息回写信息

8、并发测试

并发是指同样的功能操作不同操作员同时进行，如：不同操作员修改相同单据或参照相同单据，对于条形码测试并发主要是多个扫描终端参照的相同编号的业务单据生单并上传，应该只有一方成功；另外，扫描终端参照后台单据在保存前，后台单据被修改，如：单据被变更保存，被其它单据参照生单，被其它操作员弃审等，此时扫描终端保存时会判断单据是否被修改，判断依据是单据的时间戳，如果保存时数据库中的最新时间戳与之前参照时取到的时间戳不同，则不允许保存，应有正确提示。

七、总结

无论哪种 ERP 软件，其最终目的是满足用户的需求，因此用户的实际使用场景是我们测试的重点。条形码应用模式有三种，本文主要针对第三种应用方式测试进行重点说明。条形码功能操作虽然简单，但其数据接口影响范围比较大，涉及的单据有 scm 和制造的多个相关单据，重点需要关注量的变化。另外，水平产品使用的客户是多样的，正向和反向操作都有可能存在，因此这方面也是测试的重点。

【淘测试专栏】测试计划和需求变更

作者：季哥

大家做项目或产品过程中，遇到最多的，也觉得最有风险的就是测试计划。关于如何做测试计划，每个人都有自己的经验，让我们来看看国外的大师 James Bach 是如何制定测试计划的，希望对大家将来制定测试计划和测试策略有用。而且关于测试计划的大部分问题我想都可以在这里找到答案。

记得多位测试大师说过在项目前期我们需要制定测试计划，测试计划对于很多公司和团队来说，都有很多不同的模板，根据自己公司的特点定制化一份有效且完整的测试计划模板是非常重要的事情。一般情况下，是由项目的测试负责人来制定测试计划，制定测试计划的过程也是测试负责人了解和理解产品架构和设计的过程。 我们在乎的不是测试计划这样的一份文档，在乎的是制定测试计划的过程，过程让大家感觉更可靠和更完美。

制定一个测试计划的过程，是的确考验测试负责人的水平和全局观念，但是这里需要强调的是测试计划和测试策略不是一成不变的，在项目的开发周期过程中，测试计划可以合理的变化，测试经典可以合理的调整，从而适应不同的管理要求。下面会详细的介绍整个制定测试计划的过程，主要适用于传统软件行业，对于互联网行业，完全可以定制化一份适合自己的测试计划过程和模板。这里需要强调的是制定测试计划的一个核心就是测试范围的确定，明确的确定测试团队将测试什么，不测试什么。尤其是不测试什么的圈定，需要测试策略，管理经典，市场需求，组织决定等多个因素的衡量，也是最考验测试人员的责任心和大局观。

制定测试计划

1、分析产品

- 分析什么：

  用户（他们是谁，他们做什么的）

  操作（这个操作是干什么用的）

  产品结构（代码，文件，等）

  产品功能（这些功能是干什么用的）

  产品数据（输入的，输出的，状态，等）

  平台（外部的硬件和软件）

- 怎么分析：

  走一下产品/原型的主要流程

  评审产品和项目文档

  咨询设计人员和用户

与类似的产品做比较

- 可能的工作产出：

  产品的功能范围概要

  注释性的文档

  产品的问题列表

- 执行状态检查：

  设计人员有没有确认以及批准了产品的功能范围概要？

  设计人员有没有认为你已经正确理解了这个产品？

  你能不能将这个产品形象化并且预测正确的行为？

  你能不能造出产品的测试数据（输入和结果）？

  你能不能配置和操作这个产品？

  你有没有理解这个产品是怎么样被使用的？

  你有没有注意到设计中的漏洞或不一致的地方？

  关于这个产品你还有没有未解决的问题？

  2、分析产品的风险

- 分析什么：

  产品受到的威胁

  产品的易受攻击的地方

  失败的方式

  失败后的影响

- 怎么分析：

  评审需求和规格说明书

  评审出现问题的一些事件

  咨询设计人员和用户

  通过探索性风险分析和质量判据列表来评审产品

  识别基本的错误/失败方式

- 可能的工作产出：

  组件风险列表矩阵

  失败模型概要

- 执行状态检查：

  设计人员和用户有没有对风险分析达成一致？

  你有没有发现所有的重要的问题，而这些问题是否在测试过程出现呢？

  你是否知道在哪些地方要集中测试精力并获得最大的效率呢？

  设计人员有没有做一些事情使得重要的问题更容易的发现，或减少其发生的

概率呢？

如果你的风险分析是正确的，你是怎么发现的呢？

3、设计测试策略

- 基本策略：

  Domain testing（包括边界值）

  用户测试

  压力测试

  回归测试

  Sequence testing

  State testing

  基于文档的测试

  结构化测试（单元测试等）

- 怎么计划：

  对于风险和产品功能匹配策略

  将特殊的和实际的策略形象化

  分析是否可用自动化的机会

  使用原型去测试 probes 和 harnesses

  不要强加计划，让测试人员自己决定

- 可能的工作产出：

  各个类型的报告怎样应用的测试策略文档

  风险/任务的 matrix

  已选择的策略中存在的问题或挑战列表

  对产品覆盖比较少的部分提供的建议

  测试用例（如果是必须的）

- 执行状态检查：

  设计人员对这个测试策略达成一致了吗？

  这个策略对于项目每个参与人员以及协助人员都有用吗？

  这个测试策略是否很基本了？是否也容易的应用到这个产品中？

  这个测试策略是否透露了所以的重要的问题

4、计划安排

- 安排的内容：

  测试时间的评估和计划

  易测性的工程分析

  测试团队人员（详细的能力）

测试人员的培训和监督
测试人员的任务的指定
产品开发信息的收集和管理
项目会议，沟通，协调的方式
与其他已存在的功能之间的关系，包括开发过程中
测试平台的认购和配置
测试工具盒自动化
需要用到的测试桩和 mock
测试套的管理和维护
建立和输出协议约定
测试周期管理
问题报告系统和约定
测试状态报告的约定
代码冻结和增量测试
测试后期的压力管理
项目阶段输出协议约定
测试效率的预估

- 可能的工作产出：

问题列表
项目风险分析
任务和责任 matrix
测试时间表
与开发之间的约定和协议

- 执行状态检查：

这个项目所列的安排是否支持测试策略？
是否存在一些问题会阻碍测试的执行？
在可见性的问题面前，这些安排和策略是否适合？
你现在是否开始测试还是以后整理剩下的问题？

5、分享计划

- 分享的方式：

让设计人员和股东都参与到整个测试计划的制定过程中
更主动的获取关于测试计划的意见
尽最大可能帮助开发人员保持进度
帮助开发人员理解他们做什么会影响测试

与技术支持和写技术文档的人分享产品质量信息

让设计人员和开发人员评审并且批准所以相关的文档

记录并加强与开发之间的约定

让参与人员评审测试计划的细节

在测试计划中尽量减少没必要的信息以增加评审的效率

- 目标：

对于测试过程达到一致的理解

对于测试过程达到一致的承诺

人员对于测试过程有个合理的参与度

对于测试过程，在管理上有个合理的期望

- 执行状态检查：

整个项目团队对于测试计划有没有足够重视？

整个项目团队，特别是一线的管理层有没有完全理解测试团队的作用？

整个项目团队，有没有感觉到测试团队在项目过程中有最大的效益？

测试团队和项目其他团队有无争议性或建设性的关系？

项目团队里有没有成员感觉测试人员经常越出常轨，而不是关注重要的测试任务？

需求变更

在我们的项目过程中，特别是互联网领域，项目测试过程中，不管是前期还是后期，都会存在需求变更的问题，这些需求变更如果出来不当的话，就会影响测试质量，那么如何最大限度的降低需求的变更对测试质量的影响？

注意看这里面的几个关键词：一个是“最大限度的降低”，另一个是“需求的变更”。最后一个是“测试质量”。我们要解决好这个问题，必须要很好的理解这个几个关键词的内在含义。我想并不是所有人都能很好的理解这些术语。

那什么叫需求的变更呢？顾名思义就是项目需求发生了变化与修改（先不谈什么原因），对应测试这边测试需求也就相应的变化。这里可不要忘了一个重要的时间点，那就是一旦 PRD（产品需求规格说明书）评审通过后。其后任何一个时间点，不管是 UC（User Case，单个特性的设计需求）设计还是测试设计或是什么，只要需求发生了变化，这都属于需求的变更。

那什么叫测试质量呢？这个概念比较大，一般我们讲软件质量，这个就与我们测试人员的工作职责相关的。而对于测试质量，个人认为包括两大块，一个是测试各个阶段的产出的高质量，一个是测试各个阶段的控制的高效性。解释一下第一个是各个阶段的产出的文档的高质量，这里面包括文档的规范性，完整性，正确性，统一性。第二个是各个阶段的进度控制和项目管理的高效率。包括测试

目标以及发现缺陷，甚至是缺陷预防的持续改进等。

想必大家都知道需求变更不是个好东西吧，那我们就要想办法减少需求变更。首先需求变更往往是不可避免的。通常是项目负责人员花费了大量的气力避免需求变更，可最后需求变更总是会出现。在需求并更发生之前尽量减少需求变更，以将需求变更带来的风险降低到最低。因此，在需求人员或产品经理同用户代表或用户部门主管人员接触时，就应该向他们挑明态度，和他们协商好，特别是应该让他们清楚软件的定价应该与软件的功能相关，以及需求随意变更所带来的风险的承担者应该由客户和项目开发者共同承担。简单说让客户明白减少需求变更的重要性后，需求分析人员应该采取合适的方法同客户交流，帮助他们明确他们的需求。

还有一个就是我们要规范需求文档，需求文档应该按照一定的格式和规范来写，而且应该具备完整性、一致性、基线控制、历史记录等特性。需求变更发生后，也应该生成相应的文档，并且这些文档的书写也应该采用规范的形式来写。

前面说到得是减少需求变更的方法，这个不是一个人能做到的，是整个项目组成员共同努力的。正如前面所说，需求变更不可避免的会发生，那么当需求变更发生后我们测试人员应该如何应对呢？一般来讲，需求的变更通常意味着需求的增加，需求的减少相对很少，而且处理也比较容易。而且发现现在开发人员和 PM 对需求变更起主导作用，同时需求的变更并不能实时反馈到测试人员。这就需要流程的监控了，之前也说了一旦出现什么样的需求变更，就相应的走需求变更的流程（相信这里应该充分考虑测试人员的参与度）。充分的与开发人员沟通加上 SQA 的实时跟踪也许会减少需求的变更对测试质量的影响。

我们说到了对测试质量的影响，那么有哪些呢？一个测试设计与测试用例的文档的修改；一个是与开发沟通需求变更的成本；再一个是测试人员重复测试执行的成本。另外最关键的是由于需求变更带来的测试覆盖率的风险，特别是在测试执行后期阶段，时间计划都已经安排的很合理，这时由于需求变更，其影响可想而知。

最后总结下该如何最大限度的降低需求的变更对测试质量的影响：

- 减少需求变更
- 规范需求文档
- 与开发及时沟通需求问题
- 需求变更流程控制执行到位
- 尽量避免需求变更在测试后期阶段（成本大，风险大）
- 及时采取办法应对需求变更（时间延期，覆盖率，评估需求变更）

我们也知道需求变更的不好的影响，一个最大的影响就是需要修改测试用

例，面对这样的测试用例修改，我们该如何处理呢？首先要强调的是这样的测试用例肯定需要修改，但是不一定就要做测试执行阶段全部完成。另外一个我们可以改变写测试用例的方式和策略，这样在需求变更的时候可以最大化减少修改的地方。

当然，我也知道，大部分情况是说的容易，做起来难，确实前面说到的几个办法都是比较难做到的，但是如果我们把利益传递和分析给相关的人，比如我们的 PM、开发、SQA、需求方、经理。我相信我们是可以做一些事情来提高整个产品的测试质量的。当我们把这些做事情的方式形成了一种习惯，我们就可以享受快速和便捷的测试管理方式和问题解决方式。

使用 HttpWatch 来辅助 QTP 自动化测试

作者：文青山

摘要：本文介绍了 HttpWatch 提供的自动化测试接口的调用方法，并介绍了如何在 QTP 中具体实用的方向以及应用的实例。

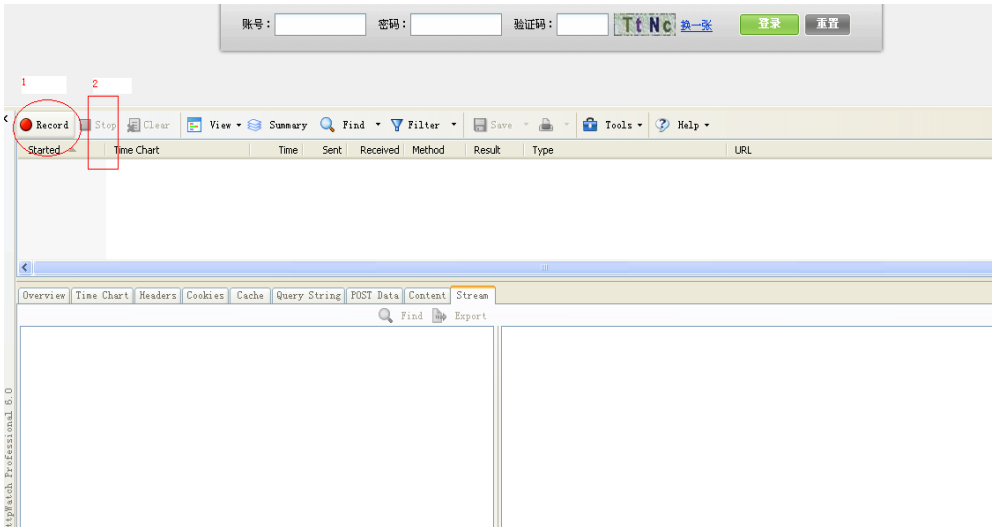
关键词：自动化、QTP、HttpWatch

一、HttpWatch 是什么

HttpWatch 是一款强大的网页数据分析工具，安装后集成在 IE 或 Firefox 工具栏中。它实现了在不改变浏览器和网络设置的基础上捕捉底层 Http 和 Https 数据，同时还能统计客户端发送请求到服务器返回请求的时间，并提供了记录日志的方法。同时还具有完备的 COM 接口，用于给用户通过编程的方式操纵 HttpWatch。

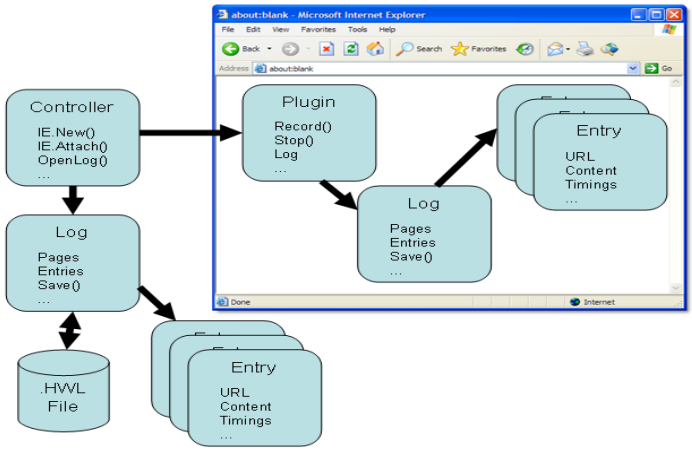
二、HttpWatch 的基本用法

点击如图中的 Record 按钮后，HttpWatch 就会记录用户在系统上的操作，并返回软件设定的相关 Http 底层数据，点击如图 Stop 按钮后，HttpWatch 将会停止记录。并在软件中列出相关 Http 流的数据信息。



三、HttpWatch 提供给自动化测试的接口

Httpwatch 自动化对象的结构图在其帮助文档中有如下的介绍：



Controller 类用于创建一个新的 HttpWatch 的接口或者打开某个日志文件，它支持 IE 或 FireFox 去创建或附加到新的接口。当使用 New()或 Attach()方法之后，浏览返回一个 Plugin 类的对象，这个对象可以控制 HttpWatch 这个软件去记录或停止或保存等相关操作，并且可以使用 GotoURL()方法加载相关页面到浏览器。Log 属性可以用来维护 Plugin 对象的日志，并可以用来激活相关日志文件里的信息，并且可以通过该属性访问 Entry 类中的一些属性，可以获取如 headers，cookies，content 等信息。

从上面的对象结构图和文字信息的描述，HttpWatch 的自动化接口，主要使用以下类和属性：

Controller 类：Controller 类总是控制的起点，它用于创建 Httpwatch 插件接口或者读取 log 文件,通过 IE 属性和 Firefox 属性的 new()方法或者 Attch()方法返回 Plugin 实例对象。

Plugin 类： Plugin 类是 Httpwatch 与浏览器交互的接口。它提供了方法用于开始和停止 http 流量的记录。还有很多方法和属性用于管理 Httpwatch 的日志文件和配置自动化记录。

Log 类： Plugin 对象的 Log 属性是 Log 对象的入口。它包含了一系列通过 Httpwatch 记录的日志。Log 对象中具有很多重要的类，比如 Pages，Entries 类。

Entry 类：每个 Log 对象都包含一组 Entry 对象，它代表的是单个 HTTP 事务的详情。

HttpWatch 提供的该自动化 COM 接口的名称为 HttpWatch.Controller。并在其帮助文档中列举了，供 C#、Ruby、JavaScript 的自动化测试接口。

如在 C#中使用的例子：

```
HttpWatch.ControllerClass ct = new HttpWatch.ControllerClass();
HttpWatch.Plugin plugin = ct.IE.New();
```

```
plugin.Record();// Start recording
plugin.GotoUrl("http://www.example.com");   // Go to web page
ct.Wait( plugin, -1);// Wait for the page to load
plugin.Stop();// Stop Recording
plugin.Log.Save(@"c:\mylogfiles\mylogfile.hwl");
plugin.CloseBrowser();
Console.WriteLine( "The log file contains " + plugin.Log.Entries.Count + "
entries");
```

当然，当使用 WatiN 自动化测试框架的时候，我们可以更好的使用 HttpWatch 提供的自动化测试接口，如下面的例子，例举了如何在 WatIN 框架中使用 HttpWatch 的例子：

```
//新建 IE 实例
IE ie = new IE();
//建立 Httpwatch 控制类
HttpWatch.Controller controller = new HttpWatch.Controller();
//将 ie 实例附加到 Httpwatch 接口，从而使 Watin 和 httpwatch 与同一
//IE 建立关联
HttpWatch.Plugin plugin =
controller.IE.Attach((SHDocVw.IWebBrowser2)ie.InternetExplorer);
/Httpwatch 开始记录
plugin.Clear();
plugin.Record();
//导航到百度
ie.GoTo("http://www.baidu.com/");
//搜索“watin”
ie.TextField(Find.ById("kw")).TypeText("Watin");
ie.Button(Find.ById("su")).Click();
//停止 httpwatch 记录
plugin.Stop();
//创建 Log 实例
plugin.Log.Save(@"c:\mylogfiles\mylogfile.hwl");
```

四、在 QTP 中如何使用 HttpWatch 提供的自动化测试接口 HttpWatch.Controller

由于 QTP 支持 VbScript 的语言，所以我们可以使用 CreateObject()的方法

创建一个对象，然后调用 HttpWatch.Controller 该接口中的方法。

如下面的实例演示了如何在 QTP 中使用该自动化接口，并通过相关方法，返回导航到百度加载完全所花的时间以及其状态码。

```
Dim URL'存放待测试的地址
URL=" www.baidu.com "
'从 Com 中创建一个 control 对象
Set control=CreateObject("HttpWatch.Controller")
'再创建一个 plugin 对象
Set plugin=control.IE.New()
'设置是否开启日志过滤器
plugin.Log.EnableFilter(false)
'清除历史 HttpWatch 的记录
plugin.Clear()
'开始记录 Http 底层数据
plugin.Record()
'导航到指定的浏览器地址
plugin.GotoURL(URL)
'等待浏览器加载完全
control.Wait plugin,-1
'停止记录 Http 底层数据
plugin.Stop
'Entries，从日志中获取所有从服务器返回的请求数据
'Summary，从页面中返回的所有服务器的数据
Set sumary=plugin.Log.Entries.Summary
Msgbox "从浏览器发出请求到服务器返回数据的时间为："&sumary.Time&"秒"
'Msgbox "从服务器返回的错误数："&sumary.Errors.Count
'
'获取返回的状态码
Set  statuscode=plugin.Log.Entries.Item(0)
Msgbox "返回的状态码："&statuscode.StatusCode
```

五、在 QTP 中实际应用的方向

1、验证链接的连通性

检查其返回的状态码来判断是否链接成功，抛弃使用检查点的方式来判断，这样的话判断结果将会更准确，避免 QTP 检查点所选的界面元素被更改时导致

QTP 报错。

如实例：

```
Dim URL'存放待测试的地址
Dim linkStatue '' 链接所返回的状态码
URL="www.baidu.com"
'从 Com 中创建一个 control 对象
Set control=CreateObject("HttpWatch.Controller")
'再创建一个 plugin 对象
Set plugin=control.IE.New()
'设置是否开启日志过滤器
plugin.Log.EnableFilter(false)
'清除历史 HttpWatch 的记录
plugin.Clear()
'导航到指定的浏览器地址
plugin.GotoURL(URL)
'等待 2 秒
wait(2)
'开始记录 Http 底层数据
plugin.Record()
''录制或用描述性编程生成点击百度首页中的新闻链接
Browser("title:="&"百度一下，你就知道").Page("title:="&"百度一下，你就知道
").Link("name:=新 闻").Click
'等待让页面加载完全
control.Wait plugin,-1
'停止记录 Http 底层数据
plugin.Stop
'获取返回的状态码
Set  statuscode=plugin.Log.Entries.Item(0)
linkStatue=statuscode.StatusCode
If 400>linkStatue Then
   Reporter.ReportEvent micPass,"返回状态码是否正确","返回状态码正确，为：
"&linkStatue
else
   Reporter.ReportEvent micFail,"返回状态码是否正确","返回状态码错误，为：
```

```
"&linkStatue
    End If
    ''关闭浏览器窗口
    plugin.CloseBrowser()
    ''回收对象
    Set control=nothing
    Set plugin=nothing
    Set statuscode=nothing
```

注意状码码为 2XX 或 3XX 时是正确的返回码，4XX 或 5XX 是错误的状态码，所以程序中判断条件是“400>linkStatue”，运行完后点击结果可见以下内容：

Step Name: 返回状态码是否正确

Step Passed

Object	Details	Result	Time
返回状态码是否正确	返回状态码正确，为：200	Passed	2011-5-11 - 20:55:03

2、检查某重要功能的传输速度和时间

在某些关键业务操作时，我们可以通过此方法来判断此功能的反应时间，起到单个用户性能测试的作用。如，某个功能的功能测试用例要求，执行该功能所耗时间不得低于 10 秒。

QTP 在某个页面操作时间很长时，并且该页面存在一些 js 对象没有回收的现象时，QTP 在该页面运行将会非常的慢，这时我们可以使用该方法来判断，如果界面超出某个时间范围则停止脚本。

如实例：

```
Dim URL'存放待测试的地址
Dim linktime''链接所花费的时间
URL="www.baidu.com"
'从 Com 中创建一个 control 对象
Set control=CreateObject("HttpWatch.Controller")
'再创建一个 plugin 对象
Set plugin=control.IE.New()
'设置是否开启日志过滤器
plugin.Log.EnableFilter(false)
```

```
'清除历史 HttpWatch 的记录
plugin.Clear()
'导航到指定的浏览器地址
plugin.GotoURL(URL)
'等待 2 秒
wait(2)
''录制或用描述性编程生成点击百度首页中的新闻链接
Browser("title:="&"百度一下，你就知道").Page("title:="&"百度一下，你就知道
").Link("name:=新 闻").Click
'开始记录 Http 底层数据
plugin.Record()
''录制或用描述性编程生成搜索某关键字的功能
Browser("百度新闻搜索——全球最大的中文新闻平台").Page("百度新闻搜索——全
球最大的中文新闻平台").WebEdit("word").Set "QTP"
Browser("百度新闻搜索——全球最大的中文新闻平台").Page("百度新闻搜索——全
球最大的中文新闻平台").WebButton("百度一下").Click
control.Wait plugin,-1
'停止记录 Http 底层数据
plugin.Stop
'Entries，从日志中获取所有从服务器返回的请求数据
'Summary，从页面中返回的所有服务器的数据
Set sumary=plugin.Log.Entries.Summary
linktime=sumary.Time
If 10>linktime Then
Reporter.ReportEvent micPass,"所用时间是否少于 10 秒","所用时间少于 10 秒，具
体为："&linktime
else
Reporter.ReportEvent micPass,"所用时间是否少于 10 秒","所用时间多于 10 秒，具
体为："&linktime
''退出 action
ExitAction
End If
''关闭浏览器窗口
plugin.CloseBrowser()
```

```
''回收对象
Set control=nothing
Set plugin=nothing
Set sumary=nothing
```

3、其它方向

比如通过获取某个功能的返回的字符串，来检查功能的正确性，或者记录 Http 流的相关内容，作为分析数据的日志等等，这些需要大家多多尝试，在此就不介绍了。

六、一个在 QTP 中使用 HttpWatch 自动化接口来进行的实例

从（五）中列举的的例子可以看出，我们如能充分利用 HttpWatch 提供的自动化接口，可以改变自动化测试常用的判断方式，使其判断方式能够更好地基于 Http 返回的数据，而不是基于界面元素。

如果你细心的话，可以发现，上面所有的例子，都只是创建了一个新的 control 对象后才开始的，显然不附合我们在自动化测试中要广泛使用的原则，那么有没有办法可以附加到的 IE，以便一个脚本多次使用呢？

值得欣慰的是 HttpWatch 提供了 Set plugin=control.IE.Attach(pBrowser) 的方法来附到一个 IE。HttpWatch 帮忙文档介绍了如何在 WatiN 中使用此方法的附加 IE 的方法，但是没有演示如何在 QTP 中使用的方法，下面的实例演示了如何使用 internetexplorer.application 接口，在一个 QTP 脚本中多次创建 control 对象，并使用 plugin.Record()和 plugin.Stop 来进行自动化测试。

```
Dim URL'存放待测试的地址
Dim linktime''链接所花费的时间
Dim linkStatue '' 链接所返回的状态码
URL="www.baidu.com"
''创建 internetexplorer.application 的一个对象，并设置为 true，默认为 false，以使 control 对象能附加到此 IE 上
Set ieBrowser=CreateObject("internetexplorer.application")
ieBrowser.Visible=true

'从 Com 中创建一个 control 对象
Set control=CreateObject("HttpWatch.Controller")

''将 HttpWatch 附加到这个 IE 上
Set plugin=control.IE.Attach((ieBrowser))
```

```
'设置是否开启日志过滤器
plugin.Log.EnableFilter(false)
'清除历史 HttpWatch 的记录
plugin.Clear()
'导航到指定的浏览器地址
plugin.GotoURL(URL)
'等待 2 秒
wait(2)
''录制或用描述性编程生成点击百度首页中的新闻链接
Browser("title:="&"百度一下，你就知道").Page("title:="&"百度一下，你就知道
").Link("name:=新 闻").Click
'开始记录 Http 底层数据
plugin.Record()
''录制或用描述性编程生成搜索某关键字的功能
Browser("百度新闻搜索——全球最大的中文新闻平台").Page("百度新闻搜索——全
球最大的中文新闻平台").WebEdit("word").Set "QTP"
Browser("百度新闻搜索——全球最大的中文新闻平台").Page("百度新闻搜索——全
球最大的中文新闻平台").WebButton("百度一下").Click

''让页面加载完全后才做下一步操作
control.Wait plugin,-1

'停止记录 Http 底层数据
plugin.Stop
'Entries，从日志中获取所有从服务器返回的请求数据
'Summary，从页面中返回的所有服务器的数据
Set sumary=plugin.Log.Entries.Summary
linktime=sumary.Time

If 10>linktime Then
Reporter.ReportEvent micPass,"所用时间是否少于 10 秒","所用时间少于 10 秒，具
体为："&linktime
else
```

```
    Reporter.ReportEvent micPass,"所用时间是否少于 10 秒","所用时间多于 10 秒，具体为："&linktime
    End If


    ''###
    ''###注意，不能再次使用 plugin 对象，否则记录的数据为历史数据，也就不正确了
    ''###
    ''再次创建 plugin2，用来记录点击网页链接所返回的时间

    ''回收已经使用过的对象
    Set plugin=nothing
    Set sumary=nothing

    ''将 HttpWatch 附加到这个 IE 上
    Set plugin2=control.IE.Attach((ieBrowser))

    plugin2.Log.EnableFilter(false)

    plugin2.Record()
    ''点击网页链接
    Browser("百度新闻搜索——全球最大的中文新闻平台").Page("百度新闻搜索_QTP").Link("网页").Click

    ''让页面完全加载完全后才做下一步操作
    control.Wait plugin2,-1
    plugin2.Stop()

    Set sumary2=plugin2.Log.Entries.Summary
    linktime=sumary2.Time

    If 10>linktime Then
    Reporter.ReportEvent micPass,"第 2 次测试所用时间是否少于 10 秒","所用时间少于 10 秒，具体为："&linktime
```

```
    else
    Reporter.ReportEvent micPass,"第 2 次测试所用时间是否少于 10 秒","所用时间多于 10 秒，具体为："&linktime
    End If

    ''关闭浏览器窗口
    ''如果我们想打开 HttpWatch 看看相关 Http 数据的话，下面这段代码就要注释掉，让网页不关闭，这样的话我们可以手动打开 HttpWatch，并能查看到相关信息和内容
    plugin2.CloseBrowser()

    ''回收对象
    Set plugin2=nothing
    Set sumary2=nothing
```

七、注意事项

本文只粗略的介绍了在 QTP 中使用 HttpWatch 的方法，并且这些介绍也只基于 IE，关于 FireFox 和 HttpWatch 提供的自动化测试类，以及其关的详细调用方法和含义需要大家自己去尝试。另外，本文中所有的例子调试的环境为 QTP10.0，WindosXP，HttpWatch7.0.24。

估算并发用户数的方法

作者：Eric Man Wong 翻译：韩宇

一、引言

为了进行容量规划和进行性能方面的管理，正式发布产品之前往往有必要估算系统能够承受的最大并发用户数。因为系统资源的使用直接与并发用户数挂钩。就拿 Web 应用来说，内存的使用，CPU 的利用率，服务器的进程/线程数，数据库连接数和网络带宽占用率都是关于并发用户数的增函数。

尽管知道并发用户数的重要性，我们还是经常通过第六感或者是大胆臆测去估计这个数值，十分缺乏理性。在本文中，我们会尝试去介绍一种简单的方法来得出这个并发用户数的估计值——通过某些其他的参数，这个值将会更加易于估算并且更加合理。

二、一种不令人满意的方法

人们时常用的一种估计方法是这样的：假设并发用户数等于全部用户数乘以某个比例。这不是一种好方法，因为就算有时候总的用户数可以可靠的估计出来，然而百分比——尽管不能总说——是一个不具有说服力的魔力数字。

必须指出的是，刚提到的这个百分比不能视为在某个时间段内登录系统的那部分用户。在某些情况下才能肯定的得出登录系统的用户数。举个例子，如果我们知道每个用户都会在每个月的某一天使用且只使用一次某个系统，那么我们可以理所当然的认为任意一天使用该系统的百分比是大约 3.3%（作者注：就是 1/30）。（译者注：30 就是提到的“每个月”）。

尽管如此，仅仅依靠这个百分比不能用来推导出并发用户数。因为在同一天使用系统的人并不是同时使用的。有的用户可能在上午使用，有的用户可能在下午使用。

我们接下来看看一种更好的方法。

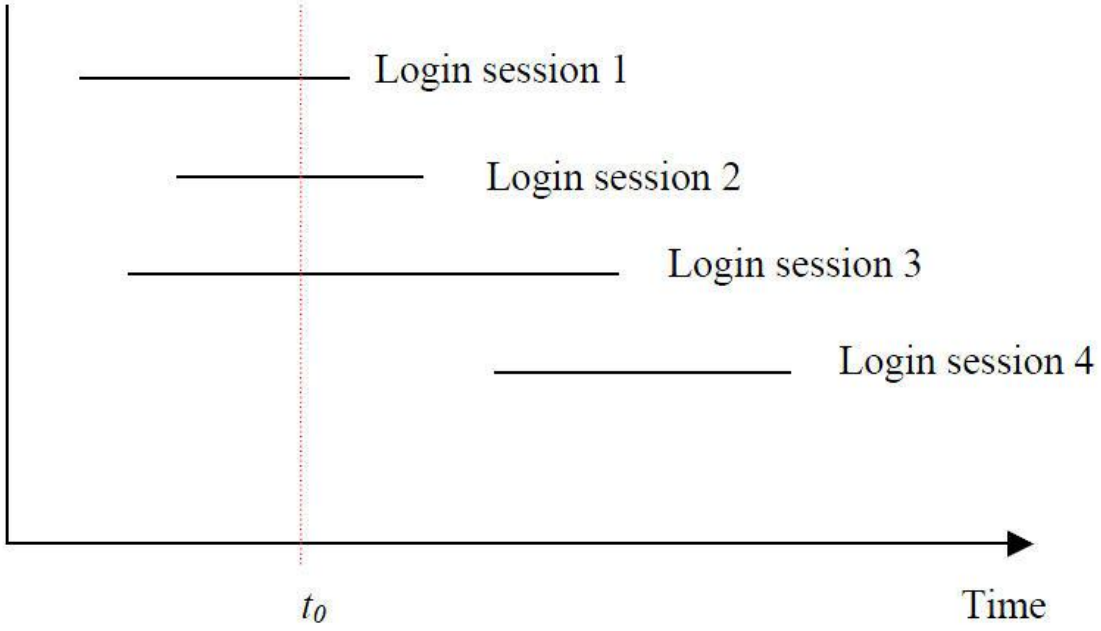
三、估算平均并发用户数的公式

我们通过定义并发用户数来开始这一节。但是在之前，我们必须搞清楚 login session 的含义。

login session 的意思是通过开始和结束时间定义的一段时间。在这段时间内，系统的一个或多个资源被占用。使用任意一个需要用户登录的 Web 应用作为例子，login session 从用户登录到系统开始，到用户退出系统结束。每次用户的登录都创建了一个用户的 session（作者注：占用了系统的内存）。login session 的时长取决于开始和结束的时间。

我们现在做好定义并发用户数的准备了。我们应该同意某个瞬间的“并发用

户数”是这个瞬间的 login session 的个数。在下图中阐明了这个定义：



横轴是时间轴。每一条水平线段代表了一个 login session。处垂直的那条线与水平线有三个交点，那么并发用户数是 3。

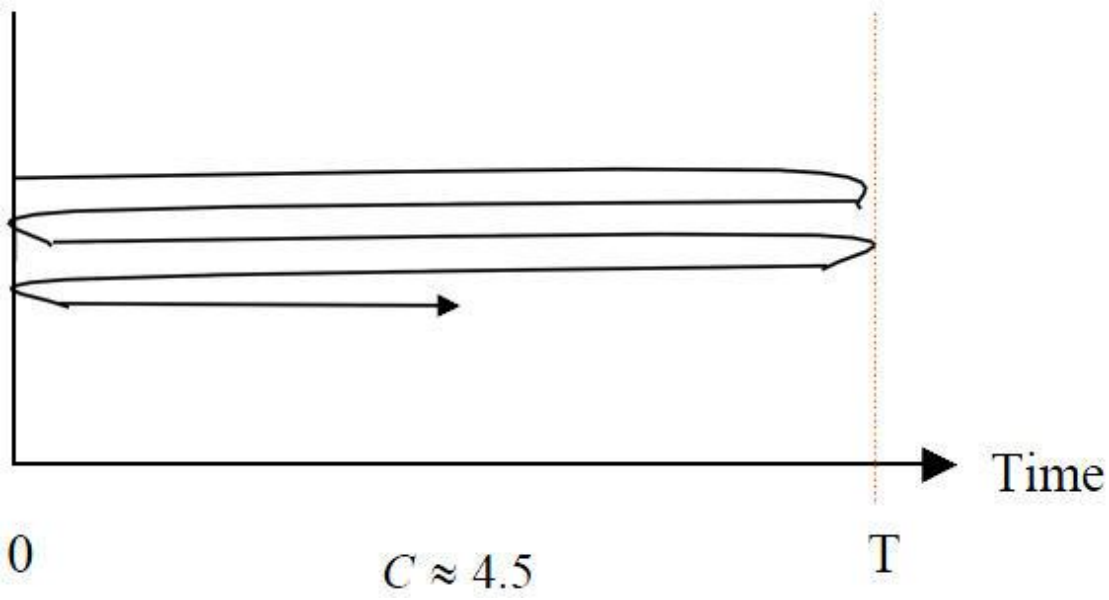
让我们关注时间从起点到随意的某个时间。下面的结论可以利用数学知识证明：

$$\text{平均并发用户数}（C）=\frac{session\text{的和}}{T} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots（*）$$

另外，如果从开始到时间 T 的范围内 login session 的总数是 n，并且平均的 login session 时长是，那么

$$C=\frac{nL}{T} \quad \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots（**）$$

正式的证明过程见附录。直觉上来看，这个公式可以这样来认为：设想每条 session 的线条都是首尾相连的。如果这条线的长度超过了，那么我们就把它折叠起来，使得每一段都长度都是T。折叠的次数就如同平均并发用户数。在下图对此做出了说明：



四、估算其中的参数

为了利用第三节的公式计算平均并发用户数（C），前提条件是在关注的时间段（T）中计算出两个参数：

- login session 的总数（n）
- login session 的平均时长（L）

在这一节中，我们会给出估算这两个参数的一些建议。

首先，必须指出通过公式得出的 C，仅仅是一个平均值。有可能并发用户数会在这个时间段内起伏很大。因此，如果我们希望利用 C 来作为并发用户数，我们应当限制关注的时间以便新的 login session 的比例（作者注：比如说 n/T 的比例）能够或多或少的保持稳定。举个例子，如果我们知道系统仅仅在工作时间内使用，我们可以只关注工作时间，而不是一整天。那么 T 就等于 8（作者注：假设 8 小时工作制），而不是 24。否则，C 的值就会大大减少，因为事实上没有人在非工作时间使用系统。

login session 的总数（n）和 login session 的平均时长（L）经常可以由用户总体和使用习惯来确定。举例来说，如果有 N 个潜在用户，并且我们知道每个用户每天可能使用系统一次、两次、三次的概率分别是 p_1, p_2, p_3 ，并且假设用户每天使用系统不会超过 3 次，那么每天的 login session 的总数就是 $N(p_1+2p_2+3p_3)$ 。另一参数——平均时长，可以通过观察样本用户来估算。

在许多的系统中，不同的用户对系统的使用频率以及平均使用系统时长波动的范围都很大。在这样的情况下，如果我们将相似的用户分类，上述的分析依然成立。我们可以计算出每一类用户的并发用户数并且将结果整合在一起。

无可争辩的是，用户的使用习惯往往是很难准确预测的。但是对于大多数的系统，尤其是内部的应用，一些这方面的合理的、粗略的数据是可以得到的。下一节的例子会对此做出说明。

五、一个例子

H 市政府准备为了其 170，000 名员工运行一个薪酬系统，员工可以通过此系统查询自己的薪酬信息。由于 IT 技能的差异、电脑的资源有限以及其他种种的原因，预计到系统上线只有 50%的员工会定期使用该系统。这些员工中的 70%会在每个月的最后一周使用系统一次。据观测，参与了用户接受度测试的员工们平均使用这个系统的时间是 5 分钟。

现在我们可以来估算一个月的最后一周的并发用户数了。我们限定关注的时间是每一天的工作时间（作者注：朝九晚五）。

$$n = 170{,}000 \times 0.5 \times 0.7 / 5（一周工作5天）$$

$$= 11{,}900$$

$$L = 5分钟$$

$$T = 8小时 = 480分钟（8小时工作制）$$

$$C = \frac{nL}{T} = \frac{11{,}900 \times 5}{480} \approx 124$$

所以，每个月最后一周系统使用的并发用户数是 124。

六、估算并发用户数的高峰值

1、理论

第三节中的公式估算了平均并发用户数。接下来很可能遇到的问题是：怎么预估高峰并发用户的数值？在本节中，我们能够看到在某些假设条件下，该高峰值也是可以估算出来的。

泊松分布是一种现成的并且广泛运用的统计模型，它适用于到达速率随机且独立的分布（作者注：该分布可以参见于大把介绍统计学的书籍）。假设在线用户数满足泊松分布，其中新登录的数量均值是λ，根据定义：

$$P（单位时间在线的用户数 = x）= \frac{e^{-\lambda}\lambda^{x}}{x!}$$

P 代表的是概率，e 是欧拉数，$x!$ 代表 x 的阶乘。

在此假设下，可以证明在任何时刻的登录成功的用户数都符合泊松分布。最惊异结论的莫过于根本不去理会保持登录的时间时长（作者注：保持登录的时长也是随机的且可以随意长短）。根据上述等式，有：

$$P(\text{任意时刻在线的用户数}=x)=\frac{e^{-c}C^{x}}{x!}$$

C 是第三节提及的平均并发用户数。（作者注：这部分的证明过于繁琐。本文将其略掉。）

众所周知，泊松分布在均值为 C 的时候可以近似的认为满足均值为 C 且标准差为$\sqrt{c}$ 的正态分布（作者注：再次说明，这些都可以在大把的统计学教材中找到）。（译者注：根据译者惨淡的统计学知识，作者漏了泊松分布要在均值大于 20 的时候才近似正态分布，不过例子中在线的平均人数是超过 20 的 ）。

如果我们令并发用户数为 X，那么就意味着 $\frac{X-C}{\sqrt{C}}$ 满足标准正态分布：均值为 0 且标准差为 1。查表可得下面的结论：

$$P(X\le C+3\sqrt{C})=P(\frac{X-C}{\sqrt{C}}\le 3)\approx 99.87\%$$

（注：正态分布表可参考 http://site.ntvc.edu.cn/jxcg/KJ/Bernoulli/N_table.htm）

通俗的讲，上面的等式意味着并发用户数小于 $C+3\sqrt{C}$ 的概率是 99.87%。这个概率对于绝大多数用途都足够大，所以我们估算高峰并发用户数为$C+3\sqrt{C}$

$$\text{高峰并发用户数}(\tilde{C})\approx C+3\sqrt{C} \quad \text{.....（＊＊＊）}$$

2、实践

在上一节，我们讲述了新登录的用户数符合泊松分布，高峰并发用户数也是可以估算的。尽管如此，在现实世界里，登陆的用户是按照以下的方式运作的：

1、沉睡状态——在非工作时间不登陆；

2、片刻状态（上升）——开始工作的时候，人们开始登录系统；在线的数量不断增加；

3、稳定期——在线的用户数保持稳定；

4、片刻状态（回落）——工作时间结束；人们退出系统；在线的人减少；

这四个状态周而复始。

像上面的这个应用系统，6.1 节的假设只适用于第 3 个状态——就是稳定期那个状态。因此，倘若我们要更精确预估高峰并发用户数的话，应该遵循下面的步骤：

1、根据经验来估计稳定期的时间段。

2、估计在稳定期的登录用户数。

3、根据第 3 节的公式（* *）计算平均并发用户数 C。

4、运用 6.1 节的公式（* * *）来计算高峰并发用户数。

通过第 5 节的一个例子来说明一下上述的步骤：

接上例，补充条件如下：假设 80%的员工进入这个薪酬系统的时间是上午 9：30 到 12：30 以及下午 2：30 到 4：30 这 5 个小时，尽管一天工作时间是 8 小时。并且，在这 5 小时之间登录数保持稳定。

$T = 5\text{小时} = 300\text{分钟}$

$n = 11{,}900 \times 0.8 = 9{,}520$

$L = 5\text{分钟}$

$$C = \frac{nL}{T} = \frac{9{,}520 \times 5}{300} \approx 159$$

$$\text{高峰并发用户数}(\tilde{C}) \approx C + 3\sqrt{C} \approx 196$$

读者可能注意到了上面的平均并发用户数和第 5 节的有差异，此处的结果更大。实际上两处的都是合理的。这正印证了第 4 节说过的，那就是——平均并发用户数很大程度上取决于所关注的时间段。在第 5 节中，我们关注的时间段是整个工作时间，所以平均值被很少人登录的那段时间所拖累了。在本节中，我们限制了关注高峰的时间段，所以结果更大。尽管两个数据都是合理的，但是后者将更加适用于展现系统的使用情况。

七、根据并发用户数得到其他有用的数据

一旦我们找出了并发用户数，其他的一些系统属性值也可以由它推出。在本节中，我们会讨论如何计算网络带宽占用率。

对于 Web 应用，请求率（作者注：比如说单位时间内的请求数，有时候视为点击率）是另一个进行容量规划的重要因素。如果可以通过样本得到用户的平均数值是 r，那么可以容易得出：

$$\text{平均总请求率}(R) = rC$$

$$\text{高峰总请求率}(R) \approx r\tilde{C}$$

式中的 C 和 $\tilde{C}$ 分别是平均并发用户数和高峰并发用户数。

拿薪酬系统的例子来说，如果平均每个用户每分钟发起 10 个请求，那高峰时平均总的请求率就大概是 1590/分钟。（作者注：）。

类似的，如果我们能够确定每位用户的平均网络带宽占用率，总共的网络带宽占用率可以用相同的方法求出。每个用户的单位时间的 bits/bytes 可以由系统通过网络传给用户的数据转换。令平均的占用率是 u ，则有：

平均总带宽占用率$(U)=uC$

高峰总带宽占用率$(\tilde{U})\approx u\tilde{C}$

八、总结

在本文中，我们提出了一个公式，利用该公式可以通过系统的登录用户数来计算某段关注的时间的平均并发用户数。另外本文还给出了一些估算这些参数的建议。

在登录用户数符合泊松分布的假设下，我们大致推断出了高峰并发用户数的上限。

文中的那几个公式并非解决找出并发用户数的灵丹妙药。它们仅仅提供了解决此类问题的一个方向。确切的公式更大程度上取决于找到用户登录的时间段以及平均在线时长，这两者又取决于用户习惯。有时候准确预估并发用户数真的是又费马达又费电。然而，我们坚信对于很多的系统来说，有些虽然粗糙但又挺合理的估算在既省时又省力的情况下是可以接受的。

附录：关于第 3 节公式（*）的证明

设 $f(t)$ 是 t 时刻的并发用户数。设想从 0 到 T 的这个时间段被平均分成了 n 个小段。每一段的时长是 $\frac{T}{n}$ 并且第 i 段在时间 $\frac{iT}{n}, i=1,2,\cdots,n$ 处结束：

当 n 很大的时候，以下对于并发用户数的估算是合理的：

$$\frac{1}{n}\sum_{i=1}^{n} f(\frac{iT}{n})=\frac{1}{T}\sum_{i=1}^{n} f(\frac{iT}{n})\frac{T}{n}$$

我们可以定义并发用户的平均数如同到那样一直延伸到无穷大。这个求和就变成了一个积分。

$$\text{平均并发用户数}=\frac{1}{T}\int_0^T f(t)dt$$

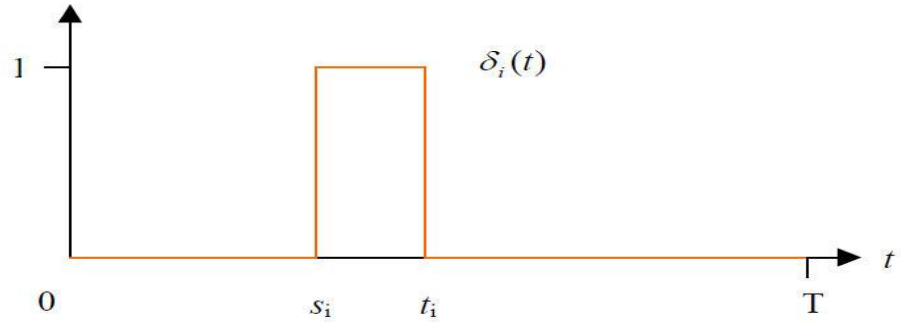
假设在 0 到 T 时刻有 m 个用户登录，分别用 1 到 m 对其编号。我们来分别考

察第 i 个用户的登录开始时刻 s_i 和退出登录的时刻 t_i 。我们再定义一个函数叫做 $\delta_i(t)$ ，它是关于第个登录用户的，定义：

$\delta_i(t) \equiv 1$，如果t在s_i和t_i之间

$\delta_i(t) \equiv 0$，其他情况

根据定义可以作图如下：



第 3 节对并发用户数的定义，可以有下面的等式：

$$f(t) = \sum_{i=1}^{m} \delta_i(t)$$

对其两边从 0 到 T 求积分，

$$\int_0^T f(t)dt = \int_0^T \sum_{i=1}^{m} \delta_i(t)dt = \sum_{i=1}^{m} \int_0^T \delta_i(t)dt$$

其中，$\int_0^T \delta_i(t)d(t)$ 显然等于$(t_i - s_i)\times 1$ ——根据上面的那个分布图。简单地说，它等于第 i 个用户的在线时长。

因此，

$$\frac{1}{T}\int_0^T f(t)d(t) = \frac{1}{T}\sum_{i=1}^{m}\int_0^T \delta_i(t)d(t) = \frac{1}{T}\sum_{i=1}^{m} 第i个用户的在线时长$$

公式（*）得证。

浅谈软件开发各阶段的自动化测试技术

作者：茹炳晟

摘要：本文根据软件开发的生命周期，依次探讨了各个软件开发阶段的自动化测试技术以及各阶段典型的测试框架与工具。使读者对软件开发各个阶段的“自动化测试”建立起全局的概念，并了解各阶段的主流测试框架与工具。

关键字：自动化测试、测试框架、单元测试、集成测试、确认测试

软件测试领域很多人都在谈论软件自动化测试，视乎自动化测试在软件测试领域是一个很时髦的词。但是绝大部分人对于自动化测试的认识还只是停留在GUI 功能测试自动化的层面。其实自动化测试本身可以根据软件生命周期的不同阶段而分为多个层面，在每个层面，“自动化”一词将具有完全不同的内涵，下面将依次根据软件开发生命周期的各个阶段，与大家分享一下各个层面的“自动化”测试的内涵以及在各层面常用的自动化测试技术与测试框架。（注：以下内容部分节选自本人作为 51testing 专家的答疑活动，可以参考 http://bbs.51testing.com/thread-313165-1-1.html）。

一、单元测试阶段的自动化测试技术

首先，在单元测试阶段，有所谓的单元测试自动化，可能很多人认为单元测试先天就是自动化的，因为单元测试会根据软件详细设计采用边界值等价类方法设计测试用例，然后以测试代码实现后再统一以自动化的方式执行。这个观点非常正确，但是这仅仅是一部分，并没有完整地描述单元测试阶段“自动化”的内涵。从比较广义的层面上讲，单元测试阶段的“自动化”内涵不是仅仅指测试用例执行的自动化，还应该包含以下五个方面的内容：

（1）单元测试用例框架代码生成的自动化。比如以下图片所示的部分框架代码就应该由工具自动生成，而不是由单元测试的开发者手工去写。单元测试开发者应该把更多的精力放在测试数据的选择与设计上，从而较大程度地提高单元测试用例的质量和用例的开发效率。

```
class TestSuite_SimpleCPPTest_cpp_4cf012d2 : public CppTest_TestSuite
{
    public:
        CPPTEST_TEST_SUITE(TestSuite_SimpleCPPTest_cpp_4cf012d2);
        CPPTEST_TEST(test_Sum_1);
        CPPTEST_TEST(test_Sum_2);
        CPPTEST_TEST_SUITE_END();

        void setUp();
        void tearDown();

        void test_Sum_1();
        void test_Sum_2();
};

CPPTEST_TEST_SUITE_REGISTRATION(TestSuite_SimpleCPPTest_cpp_4cf012d2);
```

```
void TestSuite_SimpleCPPTest_cpp_4cf012d2::setUp()
{
}

void TestSuite_SimpleCPPTest_cpp_4cf012d2::tearDown()
{
}

/* CPPTEST_TEST_CASE_BEGIN test_Sum_1 */
/* CPPTEST_TEST_CASE_CONTEXT int Sum(int, int) */
void TestSuite_SimpleCPPTest_cpp_4cf012d2::test_Sum_1()
{
    /* Pre-condition initialization */
    /* Initializing argument 1 (Num1) */
    int _Num1  = cpptestLimitsGetMaxInt();
    /* Initializing argument 2 (Num2) */
    int _Num2  = cpptestLimitsGetMaxInt();
    /* Tested function call */
    int _return  = ::Sum(_Num1, _Num2);
    /* Post-condition check */
    CPPTEST_ASSERT_INTEGER_EQUAL(-2, ( _return ));
}
```

（2）部分测试数据的自动化生成。这个主要是指工具能够根据不同变量类型自动生成测试输入数据。这个很多人都比较难以理解，因为工具是不明白代码逻辑的，它如何根据不同变量类型来生成合适的测试输入数据，并且去判断预计的测试结果呢？其实，举个例子可能就很容易明白了。比如某个被测函数的原型是 void fun（int* p， short b），那么测试数据自动生成技术就会为 int *p 自动生成空的和非空的两个指针 p 去分别执行函数 void fun（int* p， short b），并观察函数的执行情况，如果函数内部没有对空指针进行防护，那么 fun 的调用必定产生异常，从而发现函数的设计缺陷。同样的，对于 short b 会自动生成超出 short 范围的 b 去测试函数 fun 的行为。

（3）自动桩代码的生成。这个是指工具可以根据对被测试代码的扫描分析，自动为被测函数内部调用的其他函数生成可编程的桩代码，并提供基于测试用例的桩代码管理机制。此时，单元测试开发者只需重点关注桩代码内的具体逻辑实现即可。当然必要的时候，工具应该可以实现“抽桩”以适应后续的代码级软件集成测试的需求。

（4）被测代码的自动化静态分析。静态分析主要指代码的静态扫描，以识别出违反编码规则或编码风格的代码行。通常这部分工作是结合项目具体的编码规则和编码风格，由工具通过内建规则和用户自定义规则自动化地完成被测代码的静态分析。严格意义上讲，静态分析不属于单元测试的范畴，但实际上这个工作很多情况下是在单元测试阶段完成的，所以这里我也把它归入到了单元测试自动化的范畴。

（5）测试覆盖率的自动统计与分析。单元测试用例执行结束后，工具应该可以自动统计各种测试覆盖率，包括代码行覆盖率、分支覆盖率、MC/DC 覆盖率等，以此体现单元测试用例集合整体的充分性和完备性，并以此为依据增补适当的测试用例以提高测试覆盖率。

接下来列举几个满足或部分满足以上特点的典型的单元测试框架或工具。以 C/C++为例，比较主流的测试工具有 C++Test 和 Visual Unit，以 JAVA 为例有 JUnit 和 TestNG。其中 C++Test 是 Parasoft 针对 C/C++的一款自动化单元测试工具，同时也能通过“抽桩”技术而较好地应用于软件代码级集成测试阶段。Visual Unit 是 KaileSoft 推出的国产 C/C++单元测试工具，创新提出了一些全新的概念，我个人觉得还是值得推荐的。JUnit 就不多介绍了，它是一个开放源代码的 Java 测试框架，广泛应用于 Java 代码的单元测试，另外根据 JUnit 的设计思想也陆续出现了一系列的 XUnit 框架，其中比较著名的有 NUnit（针对.NET 的单元测试）、CPPUnit(针对 C++的单元测

试)、DUnit(针对 Dephi 的单元测试)、PHPUnit(针对 PHP 的单元测试)等。TestNG（Test Next Generation）是下一代的 Java 单元测试框架，它是基于 J2SE5.0 的注释（Annotations）特性的而构建的轻量级的单元测试框架，是对 JNuit 的完善与扩充，同时 TestNG 也是直接支持运行 JNnit 测试用例的。

另外，在 C/C++的嵌入式开发领域，如果单元测试用例必须运行目标系统上，那么这时单元测试自动化的内涵除了包括以上讨论的含义外，还应该包括测试用例的自动交叉编译连接以及目标系统的自动化下载与执行，这里推荐一个行业里用的比较多的工具 IBM Rational Test Real Time（RTRT)。RTRT 使用了目标适配技术（Target Deployment Port），使得测试与编译器、连接器、调试器以及目标系统结构无关，实现了跨多开发环境、多目标系统结构的自动化测试。

对于单元测试的自动化，我的感觉是入门容易精通难，你会发现，如果你有较好的开发背景，那么去学习一个单元测试框架或者工具是很容易的，但是要把这个框架很好地应用到实际项目中去就会碰到非常多的阻碍，其中包括打桩失真的处理，复杂结构的初始化，大量桩函数的管理等等。这里就不再继续具体展开了。

二、集成测试阶段的自动化测试技术

其次是软件集成测试，软件集成测试的依据是软件的架构设计和一部分模块设计。根据目前业界的情况，软件集成测试可以划分为两个不同的层面：传统意义上的软件集成测试和软件持续集成。

先看传统意义上的软件集成测试，这个从测试用例的设计与测试代码的结构来看都非常类似于单元测试，但其关注点主要是在软件模块之间的接口调用以及数据传递，相对于单元测试而言，两者最大的区别在于集成测试代码不允许打桩，必须调用真实的底层代码，单元测试代码必须打桩，以上这点就决定了集成测试“自动化”的内涵将与单元测试非常相似，尤其是在实际操作层面，比如测试用例的设计方法，测试用例代码的组织与管理，数据驱动思想的应用等等。但是很显然，集成测试对于测试框架的要求就非常高了，也就是说我们的测试框架必须可以顺利装载我们自己的并且相互依赖的软件模块，做到被测软件模块 Runnable。很不幸，据我所知目前还没有哪个测试框架能够很普适的应用于不同软件项目的代码级集成测试，所以对于软件集成测试的自动化，通常的做法是借鉴单元测试框架（比如 XUnit）的设计思想，自行开发适合于特定软件的测试框架。前段实际我就负责过一个这样的大型项目，我们自行设计并开发集成测试框架，该框架的主要功能是根

据软件架构设计依次分层加载被测模块，组织管理大量测试用例，执行测试用例的驱动代码并参数化测试输入，参数化判断测试结果，测试用例执行后的现场恢复和提供整个测试过程的 log 信息等等。另外，向像 C++Test 这样的测试工具也是适用于某些项目的软件集成测试的，前面有提到 C++Test 支持“抽桩”技术，所谓“抽桩”技术是指在单元测试用例中，我们可以人为把桩代码替换成真实的代码，显然这样的技术非常适合于软件的代码级集成测试，而且大量的单元测试用例和测试数据可以和集成测试用例复用。

软件集成测试的另一个层面就是目前比较流行的持续集成（Continuous Integration），严格意义上讲，持续集成属于软件开发实践的范畴，不属于测试的范畴，但是又和软件测试的自动化有着不可分割的关系，也就是说持续集成具有自动化测试的很多特征。我们可以使用持续集成框架/系统在非工作时间自动完成 Source code update 和 Daily Build，之后自动执行代码的静态检查，然后自动开始运行单元测试用例并统计白盒测试覆盖率，进一步地可以自动运行被测软件并执行基于 GUI 的自动化功能测试，最后将整个过程的 Log 以及相关的多份测试报告以邮件的形式通知各个干系人。

对于持续集成系统，比较著名的开源框架有 LuntBuild、CruiseControl 和 Hudson。其中 Hudson 作为后起之秀，现在已经超越了 LuntBuild 和 CruiseControl。目前企业里面应用最多的开源框架就是 Hudson。另外还有多个主流的商用持续集成系统，比如 Jetbrains 公司的 TeamCity，Atlassian 公司的 Bamboo 等。

三、确认测试阶段的自动化测试技术

最后，就是软件的确认测试阶段了，软件确认测试的依据是软件的需求规格说明书。通常软件确认测试阶段所面对的被测对象都是最终交付的软件，所以这个阶段的测试中很大一部分工作集中在基于 GUI 的功能测试和基于协议层面的性能测试。这也就是我们平时讨论最多的传统意义上的自动化测试了。

对于基于 GUI 功能测试的自动化，主流的测试工具非常多，主要有 HP 的 QuickTest Professional，IBM 的 Rational Robot 等。除此之外，比较优秀的还有基于 TDD 的 RobotFramework 和 Web GUI 自动化的 Ruby+Watir/FireWatir 和 Selenium 等等。GUI 层面的自动化测试的主要难点在于对象的识别、检查点的稳定性、测试脚本的可扩展性以及可维护性。

对于基于协议层面的性能测试，基本可以认为都是自动化的，其中最主流的就是 HP 的 LoadRunner。就性能测试而言，无论是 B/S 架构还是 C/S 架构，对于负载的发起都是基于通信协议层面的，所以测试工具的选型取决于

所选测试工具对于被测试系统（客户端和服务器）通信协议的支持。Loadrunner 可以支持绝大部分的通信协议，通常你必须了解被测系统的通信协议，然后用 Loadrunner 选择对应的协议进行试录制。通常 Socket 协议在 C/S 架构中用的比较普遍。另外，据我所知，有些公司对于 C/S 架构应用的性能测试完全采用自己开发 Load Generator 并集成性能监控程序来自动进行的。

由于时间以及篇幅有限，以上对各个阶段的每个测试框架和工具的介绍都不是很深入，错误在所难免，如果有什么地方写的不妥当，还请多多包涵。

作者简介：现任 Alcatel-Lucent 上海贝尔高级测试工程师，具有 5 年软件测试经验和 3 年开发经验。历任自动化测试技术主管、高级测试工程师等职位，具有丰富的测试框架设计与自动化测试经验。曾负责无线路由器产品的整体自动化测试方案、金融平台产品 SDK 测试框架设计、系统开发平台的白盒测试、DSP 平台自动化测试、轨道交通安全软件平台测试和多个大型 Web 项目的功能测试。

擅长领域：自动化测试、测试框架开发、白盒测试、嵌入式系统测试、测试用例设计，测试团队管理与测试流程改进。

敏捷开发中的可测试性

作者：罗斯汀

软件可测试性并不是一个新名词，但在敏捷开发中它的重要性和可行性都有一些新的变化。本文将解读敏捷开发中的可测试性。

传统的对可测试性的理解 James Bach 这样定义可测试性：“软件可测试性就是一个计算机程序能够被测试的容易程度。”维基百科则这样描述：“软件的可测试性是一个软件产出物（如：软件系统、软件模块、需求或者设计文档）在一个给定的测试环境中对测试的支持程度。可测试性不是软件产出物的内在属性，不能被直接度量。但差的可测试性会导致测试工作量的增加。在极端情况下，缺乏可测试性会阻碍软件某些部分得到充分测试甚至是根本无法被测试。”由此可见，可测试性影响测试的效率和最终产品的质量，值得关注。

我理解的可测试性可测试性虽然貌似针对测试，实际对开发的各个环节（需求、设计、编码、测试）都有特定要求。下面我们逐一探讨在敏捷开发中如何在这些环节关注和改善可测试性。

一、需求的可测试性

需求需要满足以下条件达到可测试的要求：一致、完整、无二意性、可量化（如性能需求）、在实际中在有限资源的情况下可以被验证（而不仅仅是理论上可行）。

在工作中，我们经常能碰到可测试性有问题的需求。比如：需要模拟无线网络中断的情况下手机能暂存正在发送的短信，这个网络中断就不那么好模拟。又如：一个大的需求虽然需求分析已经全部完成，但在 SCRUM 的实践中，当前 Sprint 只做其中的一部分。这部分很可能存在一些由于对后面一部分的依赖而无法测试的需求。还有一些复杂的功能或者不够友好的界面设计，因为难以理解，也就难以测试。

在敏捷开发中，需求新的表现形式为可测试性提供了更具体、可操作的方法。因为敏捷开发中，需求以用户故事的形式组织，标准的格式是“在何种前提下，何种用户希望达到何种目的。”测试因此能够更容易地识别出“测试环境（前提）”、“操作者（用户）”、“预期结果（目的）”。以往含糊的或者难以被测试的需求通过这种格式表达后更容易在早期被识别出来其不可测试性，因而可以更及时地进行调整。另外，敏捷开发强调“定义每个用户故事的完成条件”并推荐通过演示的方式来验证，这些也为可测试性提供更多的保障。还有 BDD、UADD、CADD 等各种敏捷开发的方法，提出将需求、开发、

测试这些瀑布模型中相对独立的环节在前期就进行统一考虑和安排，将可测试性的考虑也提前到了需求阶段。

二、开发过程中的可测试性

从开发的角度，敏捷开发中可以从如下方面考虑并改善可测性。

1、设计

敏捷中流行的“TDD”（测试驱动开发）强调在编码前就考虑如何测试，其中包括具体的场景和数据。这样常常能提前考虑到一些可测试性方面的困难，并尽早解决。比如：一个依靠外部接口提供输入数据的内部模块，内部开发完成后可能外部接口要稍晚才能提供，那我们如何测试或者演示呢？比较自然地，开发人员会想到预准备一些输入数据。如果输入数据的结构和内容都特别复杂，难以手工准备呢？如果各种数据类型的数据包各不相同呢？开发人员可能会在系统功能以外开发一些有界面的 mock 功能来提供更好的可测试性。这样的 mock 界面，方便开发人员也方便了测试人员，对整个团队的工作效率有很大的帮助。

有时开发人员需要改变一些程序结构的设计，为可测试性作某些折衷。例如，可测试性要求“职责单一；松耦合、低内聚；面向接口编程”等。但究竟需要支持到何种程度，还需要根据实际情况做一个平衡。

2、编码

（1）通过特定方法提供可测试性？如果一个单元与外部单元有直接的引用关系，而不修改源代码很难替代这种依赖关系。那么就需要对这个单元注入一些依赖，而不是让这个单元自己管理这种依赖。依赖注入是一种可以通过实例化它自己的外部依赖而达到去处单元责任的一种有效方式。另外，我们也常用 mock 和 fake 等方法使得代码在单元测试或者接口测试中可测。

（2）遵循编码规范，以提供可测试性在代码规范中有一些法则， 违背它的代码难以测试。如 Demeter 法则要求对远距离的对象的行为不进行直接的关联。合成/聚合复用原则要求尽量使用合成/聚合，尽量不要使用继承，否则父类发生改变，子类的行为也受到影响，需要大量的测试。

不清晰的方法名会造成类似功能对应方法混淆，干扰测试。比如，多个入口调用一个类似的功能。一个叫 save_order，一个叫 update_order，一个叫 maintain_order。当不同的流程要来串联这些基本方法进行测试的时候，人们就会很晕。我们还碰到过一个复杂的多层级的 DTO 所有对象的 oid 都叫 oid，在性能测试做参数化的时候也让人头疼不已。无论需求或者代码，可理解性是可测试性的基础。

三、测试的可测试性

从测试角度，我理解的可测试性包括两个层面：可测和易测。其中，可测包含空间（在各个层次、各个粒度都可以测试）和时间（在恰当的时间可以测试）两个维度。易测是指在可测的基础上能够方便、高效地进行测试。

1、可测

（1）空间维度

敏捷开发中强调质量由各个角色一起共同保证（实际上甚至淡化开发和测试两个角色的界限），所以无论开发人员还是测试人员需要分工协作，在不同的层次上进行不同类型和侧重点的测试。如在原子单位、在接口、在面向用户的端到端流程都需要进行测试。

（2）时间维度我理解的敏捷开发环境中，测试流程上最大的变化是要实现“边开发边测试”。为此我们需要一点安排上的特殊考虑。例如，有 2 个开发人员开发 2 个类似的用户故事。如果他们分头开发，在第 4 天同时提交测试人员测试，那么就难以将测试提前到开发过程中去。测试人员闲的时候没有东西可测，忙的时候来不及同时测大量的内容。如果我们换一种做法呢？让这两个开发人员先一起完成用户故事 1，那么 2 天后这个用户故事可以提交测试了。等到第 4 天，用户故事 2 被提交测试。这样的安排下，测试人员可以及时完成每个用户故事的测试，而且可能通过用户故事 1 的缺陷提醒了用户故事 2 在开发过程中就避免了一些类似缺陷的发生。在开发任务分配和协作的安排上，建议考虑用户故事之间的依赖关系，合理协调安排开发资源，以实现“边开发边测试”。

2、易测

（1）可观测性如果所见即所测，则直观、明了、测试效率高。KISS（Keep it simple and stupid）原则在可测试性方面仍然适用。实际中，尤其要注意考虑对 UI 不可见元素、后台程序或者测试环境受到制约的测试等。

（2）可跟踪性敏捷开发中，不希望在所有的东西都集成在一起后才发现一堆缺陷，难以定位。通过日志、 消息池等方式将事件和数据等中间结果记录下来，可以为跟踪问题提供有效帮助，从而提升了可测试性。在跟踪的时候，如果可以从隔离性的角度考虑在哪些点上记录和输出中间结果，会对定位和解决问题提供很大的帮助。这一方法在一些接口测试中尤其有用。对于一些后台的异常，也应该以 Email 等方式主动通知开发或系统支持人员，主动及时查错。

（3）将可观测性和可跟踪性结合起来前面提到的对中间结果的记录，大多是在后台静悄悄地发生，对用户并不可见，对测试人员也不直观。有时，对某些常用又复杂的分支，可以提供一些有界面的系统辅助功能，让用户自

助查找和定位问题。比如，让用户可以查看到系统接受的消息，用户可以自行判断是消息没有收到，还是收到但消息内容有错，或者我们内部程序处理除了问题。这种功能虽然需求一开始可能不会提出，但根据我们的经验，它无论是对调试、测试、实施还是系统支持都提供了极大的帮助。

（4）可控制性

1）对测试环境和测试版本的管理可控制性要求在给定输入的前提下，系统的输出是可以被稳定预测的。敏捷开发中，因为希望在更短的周期内能提供可以运行的程序，因此，对程序行为的可控制性应该做更精细的管理，比如对测试环境、测试版本等的管理方面。

例如，对测试环境定期的健康检查，保证环境总是可用。尤其是和很多系统共用一台物理服务器，或者对方系统的环境不受我方控制等情况下，环境的可用性更需要得到保障。另外，不同的测试环境，其配置上的差异也需要清晰地记录、管理和分享，使得不同类型、目的的测试能够在各个环境无干扰地独立进行。

对于版本，提倡持续集成和每日构建。将发现的问题锁定在每日的修改基础上，一旦发现问题，立马解决，不让缺陷的影响蔓延和纠缠，影响程序行为的可控制性。

2）利用自动化测试验证系统行为是否可控自动化测试的特点是能够对于预设的确定的输入验证系统是否产生了特定的输出。在这一点上，它特别符合可控制性的要求。所以我们可以通过自动化来验证被测软件的可测试性。通过自动化测试来进行模糊测试，使得原本几乎不可能大量覆盖的测试得以实现。对于已有的自动化测试脚本，可以花很少的代价在任何时间让“机器人”来帮助我们进行想要的测试，对程序的稳定性作出相应的判断。自动化的回归测试为敏捷提供了坚实保障。而压力测试工具则使得大量并发的测试能够在有限的资源条件下轻松实现。

结语

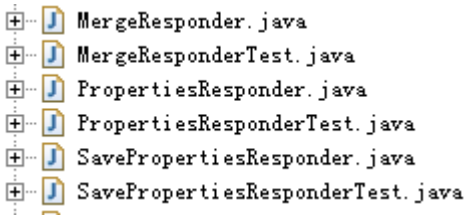
敏捷开发对软件可测试性有了更为迫切的要求，同时也有一些更可操作的实践。基于已有的理论和方法，让我们一起来思考、实践和改善软件的可测试性，达到最终改善软件质量的目标！

从 Fitnesse 中学习 Java 单元测试

作者：邵育亮

从第一次知道 Fitnesse 这个集成测试工具到现在也已经差不多有 2 年多的时间了。在这个期间把 Fitnesse 的源码也算是反反复复阅读了很多遍，算是对其实现的原理和方法有所了解。在最近一次对 Fitnesse 最新版本代码的研究中我发现，Fitnesse 除了是一个很好的开源集成测试框架之外，它的源代码还是一个非常好的实施 Java 单元测试的例好例子。

首先我们可以看到在 Fitnesse 的每个类，都有一个对应的 Test 的类来构建针对这个功能类的单元测试。



一个好的单元测试的命名是非常重要的，它可以让我们很清楚的指导我们的单元测试类是针对哪个功能类编写的。Fitnesse 的开发团队采用的命名规范是在功能类的名字后面增加 Test 的后缀，这样可以在一个包里面很好的区分功能类和测试类。

单元测试类和源码之间的放置关系也是我们经常遇到的问题。在这里 Fitnesse 的开发团队把所有的单元测试类（*.Test）和源码是放在同一个项目里面，并且也放在同一个包内，这样在功能上非常容易区分和归类。我们可以很容易的找到单元类的测试类在哪里。其次这样的单元测试类在编写上更加方便，不太收到 java 的 access 控制的影响。例如 protected 的变量在包内是可见的，这样的话同一包内的单元测试类就可以对很轻松的针对它进行单元测试。第三在 Fitnesse 的发布的 build.xml 中，也很好的继承了单元测试，这样在发布的过程中对于质量有了很好的控制，做到了单元测试自动化。参考下面的 build.xml 文件

```
<target name="unit_test" depends="compile" description="run the unit tests">
    <junit forkmode="once" fork="yes" printsummary="no" haltonfailure="yes" haltonerror="ye
        <classpath refid="classpath" />
        <formatter type="xml" usefile="true" />
        <formatter type="plain" usefile="false" />
        <batchtest todir="test-results">
            <fileset dir="src">
                <include name="**/*Test.java" />
                <exclude name="**/ShutdownResponderTest.java" />
                <exclude name="**/QueryTableBaseTest.java" />
                <exclude name="**/Test.java" />
                <exclude name="**/SystemUnderTest.java" />
                <exclude name="**/MySystemUnderTest.java" />
            </fileset>
        </batchtest>
    </junit>
</target>
```

我们到底应该对哪些方法做单元测试呢？这个问题也一直是我们在做单元测试的时候一直困扰的问题。我们之前习惯上算是针对没一个方法构建单元测试，这样的单元测试在覆盖率上很高，但是工作的时间成本就会直线上身。测试工作需要在时间和质量上寻找一个平衡。从 Fitnesse 的源代码中可以看到，Fitnesse 的开发团队并没有采用上面的那种单元测试的方法。他们采用的是针对主要的功能方法进行单元测试。我们来看下面这个例子

Response 是一个 Fitnesse 中重要的一个功能类，用来实现基于 Http 协议的关键。我们都知道 Http 的协议中一些标准的 Response Status Code，例如

```
private static Map<Integer, String> reasonCodes = new HashMap<Integer, String>() {
  private static final long serialVersionUID = 1L;

  {
    put(100, "Continue");
    put(101, "Switching Protocols");
    put(200, "OK");
    put(201, "Created");
    put(202, "Accepted");
    put(203, "Non-Authoritative Information");
    put(204, "No Content");
    put(205, "Reset Content");
    put(300, "Multiple Choices");
    put(301, "Moved Permanently");
    put(302, "Found");
    put(303, "See Other");
    put(304, "Not Modified");
```

Response 提供了一个公有的方法来获取 status code 和对应的 message

```
public static String getReasonPhrase(int status) {
  String reasonPhrase = reasonCodes.get(status);
  return reasonPhrase == null ? "Unknown Status" : reasonPhrase;
}
```

这个方法是整个 Fitnesse 服务中需要被频繁调用的，我们可以在

ResponseTest 单元测试类中针对这个方法的单元测试

```
@Test
public void reasonCode() throws Exception {
  checkPhrase(100, "Continue");
  checkPhrase(101, "Switching Protocols");
  checkPhrase(200, "OK");
  checkPhrase(201, "Created");
  checkPhrase(202, "Accepted");
  checkPhrase(203, "Non-Authoritative Information");
  checkPhrase(204, "No Content");
```

这个测试函数中调用的另外一个测试函数 checkPhrase，

```
private void checkPhrase(int reasonCode, String reasonPhrase) {
  assertEquals(reasonPhrase, Response.getReasonPhrase(reasonCode));
}
```

这样就完成了针对所有可能的 Http Response Code 的测试的覆盖。这里只是一个简单的 Fitnesse 中的例子，在很多其他的单元测试类中我们都能看到。

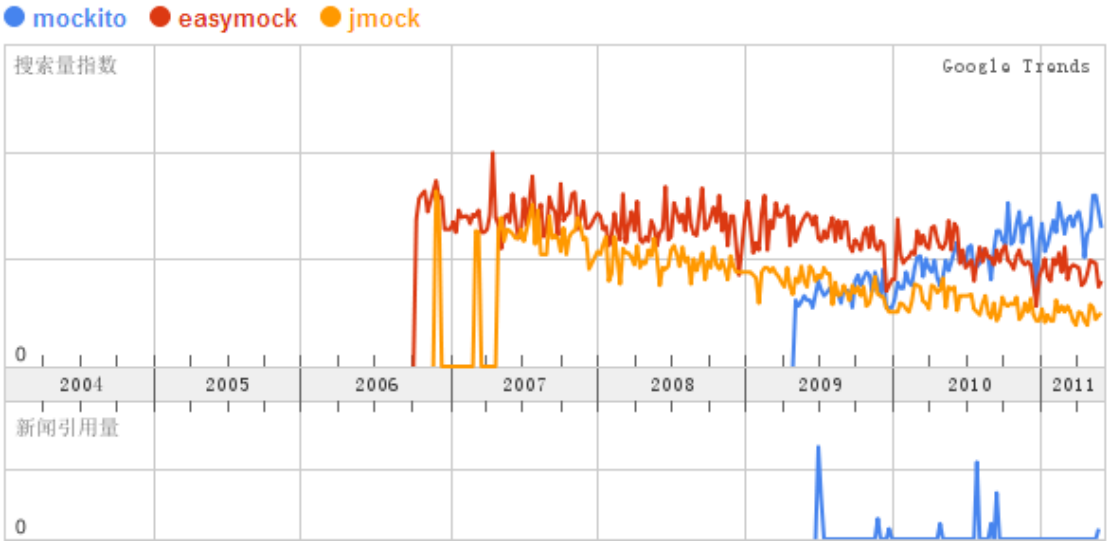
针对主要功能函数的构建单元测试对于独立进行单元测试的测试团队来说是一个很大的考验。正式因为 Fitnesse 源码的单元测试是其开发团队自己编写的，他们可以很清楚的指导哪个函数是关键函数，哪个是不重要的。但是如果是独立的测试团队，他们就很难去判断被测试函数的重要性来决定是否需要构建单元测试。

单元测试中最最重要的一个概念就是驱动模块和桩模块。从百度百科上我查到的定义如下：

驱动模块是用来模拟被测试模块的上一级模块，相当于被测模块的主程序。它接收数据，将相关数据传送给被测模块，启用被测模块，并打印出相应的结果。

桩模块：桩模块（Stub）是指模拟被测试的模块所调用的模块，而不是软件产品的组成的部分。主模块作为驱动模块，与之直接相连的模块用桩模块代替。在集成测试前要为被测模块编制一些模拟其下级模块功能的“替身”模块，以代替被测模块的接口，接受或传递被测模块的数据，这些专供测试用的“假”模块称为被测模块的桩模块。

在 java 中我们常用的用来构建驱动模块和装模块的技术有 EasyMock，JMock，相信这块很多人都已经有所了解，EasyMock 采用了 Record 和 Replay 的方式来实现针对某个类和接口的 Mock。在 Fitnesse 的单元测试中，我发现他们使用了另外的一个 Mock 工具，叫 Mockito。我现在 Google Trend 上 Google 了一把，发现原来 Mockito 目前已经在 Mock 领域被很多用户所接受，热门度超过了 EasyMock 和 JMock



Mockito 支持的语言很多，除了 Java 之外，还支持 Python ， Javascript，Scala，Perl，让我这个 Python 控真是大好消息啊。

相对于 EasyMock 这种已经被广泛使用了的 Mock 的工具，为什么像 Fitnesse 开发团队这样越来越多的人开始选择使用 Mockito，它有什么优势呢？

- 不需要采用 Recode/Replay 的模式，只需要 Verify 和 Stub 就可以了
- 非常容易的构建 Mock
- 使用 hamcrest matchers 来匹配参数

我们直接从 Fitnesse 的源码中来看如何使用 Mockito 来构建单元测试，看一下 ExecutionReportTest.class 中的测试方法

```
@Test
public void shouldHandleMissingRunTimesGraceFully() throws Exception {
  TestExecutionReport report = new TestExecutionReport();
  Element element = mock(Element.class);
  NodeList emptyNodeList = mock(NodeList.class);
  when(element.getElementsByTagName("totalRunTimeInMillis")).thenReturn(emptyNodeList);
  when(emptyNodeList.getLength()).thenReturn(0);
  assertThat(report.getTotalRunTimeInMillisOrZeroIfNotPresent(element), is(0L));

  element = mock(Element.class);
  NodeList matchingNodeList = mock(NodeList.class);
  Node elementWithText = mock(Element.class);
  NodeList childNodeList = mock(NodeList.class);
  Text text = mock(Text.class);
  when(element.getElementsByTagName("totalRunTimeInMillis")).thenReturn(matchingNodeList);
  when(matchingNodeList.getLength()).thenReturn(1);
  when(matchingNodeList.item(0)).thenReturn(elementWithText);
  when(elementWithText.getChildNodes()).thenReturn(childNodeList);
  when(childNodeList.getLength()).thenReturn(1);
  when(childNodeList.item(0)).thenReturn(text);
  when(text.getNodeValue()).thenReturn("255");
  assertThat(report.getTotalRunTimeInMillisOrZeroIfNotPresent(element), is(255L));
}
```

这个方法中使用 mockito 来定义了方法调用的返回值，在 when 中定义，当方法在第一次调用并符合参数要求的时候，就会返回 thenReturn 中的值

```
when(element.getElementsByTagName("totalRunTimeInMillis")).thenReturn(emptyNodeList);
when(emptyNodeList.getLength()).thenReturn(0);
```

在实际的 assertThat 中真正的调用测试代码，并在后面的代码调用的堆栈中实际调用了上述定义中的方法

```
protected long getTotalRunTimeInMillisOrZeroIfNotPresent(Element documentElement) throws Exception {
  String textValue = XmlUtil.getTextValue(documentElement, "totalRunTimeInMillis");
  return textValue == null ? 0 : Long.parseLong(textValue);
}
```

其中触发的调用：element.getElementsByTagName(“totalRunTiemInMIlls”); 实际返回的就是 emptyNodeList 的对象。更多的 Mockito 的使用可以看一下其官方的网站 http://mockito.org/。

越来越多的开源项目，开源测试工具让我们从中获益不少。作为一个测试工程师，在使用开源测试工具的同时，也应该去研究一下其源码。一来作为一种 Code Review 能力上的锻炼。同时从中也可以学习到单元测试的相关技巧。Fitnesse 的源码就是一个很好的基于 Junit 构建单元测试的例子。我也通过学习，更好的在自己的项目中开展了单元测试。

轻松搞定 web 兼容性测试

作者：李佳

摘要：

众所周知，对于产品化的软件，其使用对象往往是不可预知的。对于 web 应用，我们无法预计用户的客户端配置和运行环境。所以，做好兼容性测试是非常重要的。兼容性主要包括浏览器版本、显示器分辨率、操作系统版本、浏览器语言等。同一个功能，需要在以上每种情况下测试通过才算完成。为了更易于测试，我们希望不同配置和版本的浏览器窗口能够并存。本文为大家列举了解决此问题的多种办法，由浅入深，最终实现最理想的测试状态。

关键字：兼容性测试；IE、vmware；VirtualPC

正文：

对于进行产品化软件研发的企业来说，进行兼容性测试是非常重要的。因为我们创建的软件不是被某一特定用户使用，用户客户端及运行环境的多样性是不可预计的。Web 应用软件的兼容性需要包括以下几种情况：浏览器语言、浏览器版本、操作系统版本、显示器分辨率等。同一个功能，需要在以上每种情况下测试通过才算完成。这实际上就带来了测试的复杂性。测试人员希望不同配置和版本的浏览器窗口能够并存，这样才能更易于测试。可是微软对 IE 进行了限制，使这一想法无法实现。本文为大家列举了解决此问题的多种办法，由浅入深，最终实现最理想的测试状态。

方法一：IE Tester 和 IE Collection

IETester 和 IE Collection 这两款软件的产生旨在解决多 IE 版本共存的问题，均可以在网上下载，并且是免费的，是使用最广的 web 兼容性测试软件。但是在使用时存在一些缺陷：

1、不是所有机器都可以安装成功和正常使用，经常报错、死机，或导致原有的 IE 无法正常使用

2、其实现原理是模拟的某个版本 IE 的技术，显示效果与真实 IE 还是有区别

方法二：IE+DeveloperTools

自从 ie8 推出以后，微软在 IE 内置了名为 DeveloperTools 的一个小工具，通过菜单栏的"工具->开发人员工具"可以开启这个功能，快捷键为 F12。

在富客户端应用时代，前端技术突飞猛进发展，为了解决更多的客户端程序问题，此工具对于开发人员调试本地代码（例如 JS 错误），有很好帮助。并且更重要的是，它可以改变 IE 的设置，模拟不同的客户端情况进行本地调试。

切换 IE 版本：菜单栏->浏览器模式，支持 IE8，IE7，和 IE8 兼容性视图，如果是 IE9，则支持 IE7，IE8 和 IE9。

切换分辨率：菜单栏->工具->重新调整大小，支持常用分辨率和自定义分辨率

缺点：

1、无法对浏览器语言进行设置

2、没有 IE6 模式

3、显示效果与真实 IE 还是有区别

方法三：IE+DeveloperTools+Vmware's Unity 模式

即使有以上两种解决办法，但是我们还是有许多切实问题没有解决：

1、微软限制了同一操作系统下只可以装一个版本的 IE，所以无法实现真正的多版本 IE 共存

2、不同 IE 进程的设置（例如语言，Internet 选项设置）是统一的

3、不同版本 IE 无法搭配不同版本操作系统进行测试

面对这样的刚性限制，我们只能设法去运行多个操作系统，间接实现多个 IE 并存。这样就需要用到虚拟机（vmware）。

建立一个虚拟机，安装 winxp（因为 xp 预装 ie6）。可以使用精简版 winxp，因为我们不需要其他软件，只需要一个 ie，如果通过 ghost 安装更可以节省时间。做好一个虚拟机后，通过 clone 功能，复制多份。启动多个虚拟机（能同时启动虚拟机的数量，完全由你的内存决定），可以把不同虚拟机里面的 IE 设置为不同配置（分辨率、语言），实现覆盖各种测试情况。这时遇到的最大操作问题就是需要在不同虚拟机窗口之间切换以操作 IE。许多人遇到这种情况就望而止步了，感觉很麻烦。殊不知在这个问题上 vmware 的 unity 模式派上了大用场。Unity 模式使虚拟机里面的应用程序直接显示在本机桌面上，并且在任务栏上可以看到的应用如同本机应用一样。并且可以进行最小化和拷贝粘贴功能。更强大的功能是在本机可以直接操作虚拟机的“开始”菜单。同时从微软官方技术社区 MSDN 下载 DeveloperTools For IE6，使 IE6 也可以切换分辨率。这样从显示和

操作上就真正实现了“多 IE 共存”

到此，我们把需要进行的兼容性配置总结一下：

浏览器版本：IE6，IE7，IE8，IE9

浏览器语言：中文，英文

显示器分辨率：1024*768，1280*800，1280*768 等

操作系统版本：Winxp，Vista，Win7

还有一些特殊的应用和浏览器设置组合，有了此处介绍的方法，我们可以把以上情况任意组合，同时进行测试。

方法四：IE+DeveloperTools+Virtual PC’s Publish Virtual Application

此时虽然已经前进了一大步，但是仍旧还存在一个问题：需要手工启动每个虚拟机并在前台运行。我们可不可以大胆的设想一下，能否从宿主机的桌面直接启动虚拟机的应用？好，我们换一个虚拟机软件--Windows Virtual PC，win7 已经把 Virtual PC 很好的集成。同样安装好操作系统和 IE 后，使用 Virtual PC 的一大主打功能--"发布虚拟应用程序到桌面"，直接在本机桌面建立一个快捷方式，此快捷方式将直接启动虚拟机的应用(比如 IE)，同时虚拟机也将在后台启动并运行。

操作方法：手动添加快捷方式。在虚拟机操作系统中，将快捷方式从存放可执行文件的文件夹拖放到所有用户都能访问的"「开始」菜单"文件夹。例如，在 Windows XP 中，将快捷方式复制到"%systemdrive%\Documents and Settings\All Users\「开始」菜单\程序"中。 之后，宿主机就可以在「开始」菜单中"所有程序"->Windows Virtual PC->虚拟机名称下面会出现应用程序的名称。

最终，我们在本机轻松简单地搭建起了兼容性测试环境，不需要为了兼容性

测试而耗费更多的机器资源，提高了测试人员的工作效率。