自动化测试、性能测试
经验分享与交流

网易杭州研究院 钱蓓蕾
qbeilei@163.com
2009-11-28

我们的目的

- 交流工作中的心得和经验
- “互惠互利”、共同提高

今天的交流内容

- 性能测试
  - 我们目前使用的性能测试工具
  - 性能测试的步骤
  - 性能测试的经验交流
- 自动化测试
  - 我们目前使用的自动化测试工具和框架

测试工具Grinder

- http://grinder.sourceforge.net/
- 准确地说，是负载生成和性能统计的工具

测试工具grinder-选择理由

- 开源，所以不需要license
- 由于源码开放，我们碰到问题可以直接修改源码解决
- 开发团队活跃、更新频繁
- User Guide文档齐全。

测试工具grinder-选择理由

- 纯JAVA，适用于测试J2EE应用
- 应用面广，可以测试任何的JAVA代码
- 测试脚本采用Jython编写，支持各种参数化和关联操作
- 支持分布式负载，负载生成器可以运行在Windows和Linux。

性能测试步骤（一）-熟悉应用

- 这是整个性能过程最关键的步骤之一，毋庸质疑。
- 我们必须了解：应用的架构
  - 以我熟悉的应用类型为例。了解了应用架构，我们才能知道，我们需要模拟的是：一般的html静态文件请求、一般的servlet和jsp请求、AJAX请求、还是远程调用请求，等。
- 我们必须了解：应用的功能逻辑

性能测试步骤（二）-测试需求

- 我们得到的测试需求往往是这么描述的：
  - 这个系统能否支撑100万的uv（每天登录系统的人次）。
  - 言下之意是：按照目前的硬件性能和数量，系统能否支撑100万的uv。
- 然而，我们了解的是吞吐量、响应时间等指标
  - 吞吐量：系统每秒能处理的请求数，这个指标从服务器的视角，表征系统容量
  - 响应时间：从请求发出到第一个字节返回所需要的时间，这个指标从用户的视角，表征系统响应速度。

- 那么，请问开发同事：能把测试需求转化成我们熟悉的吞吐量和响应时间吗？
- 。。。。
- 答案常常是否定的。

性能测试步骤（二）-测试需求

- 怎么办：只能由我们根据经验，把100万uv转化成一系列的指标。
  - 响应时间：根据国外的一些资料，一般操作的响应时间不能高于3~5秒；重要操作，如结账操作的响应时间不能高于15秒。
  - 吞吐量：可以根据已经上线的类似产品进行估计。或者，采用80/20原则进行估计。我们经常使用的是80/20原则。

性能测试步骤（二）-测试需求

- 80/20原则，又称帕累托效应，比如，80%的社会财富掌握在20%的人手里。
- 应用于测试：从uv计算吞吐量
  - 根据80/20原则，80%的用户会在20%的繁忙时间内登陆。则繁忙时间每秒大概会有（1000000*80%）/(24*3600*20%)=50个用户登陆，也就是说，登陆操作的吞吐量是50/s

性能测试步骤（二）-测试需求

- 虽然已经有了响应时间和吞吐量指标，但是测试需求还是不明确的。
- 我们的测试目的是什么？
  - 是验证当前硬件和软件配置能否支撑100万uv？
  - 是测试当前的硬件和软件配置最多能支撑多少uv?
  - 是帮助开发寻找性能瓶颈？

- 答案往往是：都要！

性能测试步骤（二）-测试需求

- 根据我们的经验，开发的需求往往是这样的（当然开发一般不会说得那么详细，^_^）：
  - 首先，请你们验证能否支撑100万uv。
  - 如果不能支撑，请找一下性能瓶颈。
  - 主要性能瓶颈解决后，请估计能支撑多少uv，如果不到100w，请估计要加多少机器。
  - 如果能支撑100万，请再加压，看看达到300万uv的时候，系统的性能。
- 这么一细化，需求基本明确了。

性能测试步骤（三）-测试准备

- 测试准备包括测试客户端机器准备、测试数据准备、测试脚本准备。

性能测试步骤（三）-测试准备

- 客户端机器：
  - 要足够，否则，如果瓶颈在客户端，就无法评估服务端。
  - 要和服务器保持网络通畅，否则，如果瓶颈在网络，也无法评估服务端。包括：
    - 网络带宽要高于服务器吞吐量
    - 网络带宽要稳定。

性能测试步骤（三）-测试准备

- 测试数据
  - 如果被测功能涉及数据库和高速缓存，通常需要预设很大的数据量才能凸显性能瓶颈，这通常是挺困难的一个环节。
  - 如果是已经上线的应用，数据可以从线上拷贝得到；如果还没有上线，那需要构造类似于线上的数据量。
    - 比如，要测试群聊性能，我们首先需要注册大量用户；然后把测试用户都加入到聊天群中。
    - 测试数据准备的脚本，有时候比测试脚本本身还要多。
  - 对于实在没有办法构造大数据量的情况，如果要测试高速缓存，我们有时候会按数据量的比例减少高速缓存，以使测试结果尽量准确。

性能测试步骤（三）-测试准备

- 测试脚本
  - Grinder脚本用jython实现
  - 测试脚本的实现往往会花费比较长的时间
  - 因为涉及到应用实现的细节，需要和开发不断交流才能完成。这也是需要了解应用架构的原因之一。
  - 关于sleep time
    - 基于真实模拟的考虑，sleep time还是尽量按照真实时间，并给一定的偏差。
    - 不过对于测试客户端来说，sleep time往往会引起很多客户端测试线程的调度，浪费客户端系统资源。
    - Sleep time越小，客户端能模拟的吞吐量就越大，所以，实际测试中，我们往往会把sleep time设置为0 。

性能测试步骤（四）-测试执行

- 测试的执行中，需要监控测试客户端和服务器性能，监控服务器端应用情况：
  - 客户端的系统资源（cpu、io、memory）情况
  - 服务端的系统资源（cpu、io、memory）情况
  - 服务器的jvm运行情况
  - 服务端的应用情况，看是否有异常
  - 响应时间、吞吐量等指标

性能测试步骤（四）-测试执行

- 系统资源监控，linux下可以采用的工具有：vmstat、top、meminfo等。
- JVM的监控，可以用jprofiler工具，linux下面的jmap、jhat等。
- 响应时间、吞吐量等，由grinder提供。

性能测试步骤（四）-测试执行

- 上述这些信息，一般在测试结束后，均需要归档整理，已备后续详细分析
- 我们自己开发一套脚本，用于以固定的频率获取测试客户端和服务器的vmstat和top输出、grinder的log，并从中截取有用信息保存，用于事后分析。

性能测试步骤（四）-测试执行

- 每次测试运行完以后，肯定会增加很多数据，需要考虑本次执行对数据量的影响，如果数据量的变化对后续测试会有影响，则需要清理数据。

性能测试步骤（五）-测试分析

- 测试分析一般跟测试监控息息相关，在测试执行的过程中，用各种监控工具能看到系统运行的状态，并及时发现问题。
- 常见的问题有：
  - 内存问题
  - 有限资源竞争问题

性能测试步骤（五）-测试分析

- 内存问题
  - 从top中看tomcat的内存占用，这个是不准的，需要用专门的内存分析工具来查看。
  - 工具：jmap，jhat，jstat，可以得到内存快照，得到堆内存的详细信息。
  - 垃圾收集配置会影响系统性能，如果内存块生成和销毁量很大，则能看到系统吞吐量随垃圾收集呈现周期性的变化。
  - 从理论上来说，JAVA会出现内存泄漏的情况，不过我们在被测试的应用中还没有发现过这种情况。
  - 但是，在某些系统架构下，内存会成为瓶颈问题。比如我们曾经测试过聊天系统，每个长连接需要占用5M内存，那么，一台10G内存的服务器只能保持2000个长连接。

性能测试步骤（五）-测试分析

- 共享资源竞争问题
  - 有限资源的竞争有很多，比如Service层的一个共享对象，比如数据库连接，比如数据库中的某一个使用频率很高的数据表。
  - 一个共享资源在一个时间点上，只能被一个线程获得，其他线程必须等待，这就容易造成很多线程的timed wait状态。
  - 通过jprofiler工具，能够得到线程快照，并分析改进方法。

性能测试经验交流-偶然性问题

- 跟一般的功能测试一样，性能测试也会出现偶然性问题。
- 碰到这种问题，我们需要发挥测试人员的革命精神，追查到底。我们常发现的因素如下：
  - 外部因素变化，比如，某几次测试，有时候好，有时候不好，并没有规律可循。最后发现原来是因为网络不稳定造成。
  - 请求返回变化。有时候第二次请求的内容取决于第一次的返回信息（也就是所谓的“关联”），这种关联一般通过string的parse实现，而这一般都不是很可靠，返回一旦变化，可能就会出错。
  - 应用服务器如果是集群，一个用户请求某一台服务器能得到正确返回，但是如果换做另一个用户，可能该服务器并没有该用户的信息，所以返回错误。

性能测试经验交流-客户端并发

- 测试客户端要模拟高并发，必然要启动多线程，所以肯定也会存在线程并发问题。比如：
  - 在做参数化的时候，存储参数的数组就是一个共享对象。如果要使每个线程的每次循环都读取不一样的参数，那数组下标的更新需要注意并发问题。
  - 比如，如果在脚本中要调用System.out，那么也需要注意这也是一个共享对象，如果调用System.out过多，会导致线程的等待，使客户端性能降低。

性能测试经验交流-测试人员

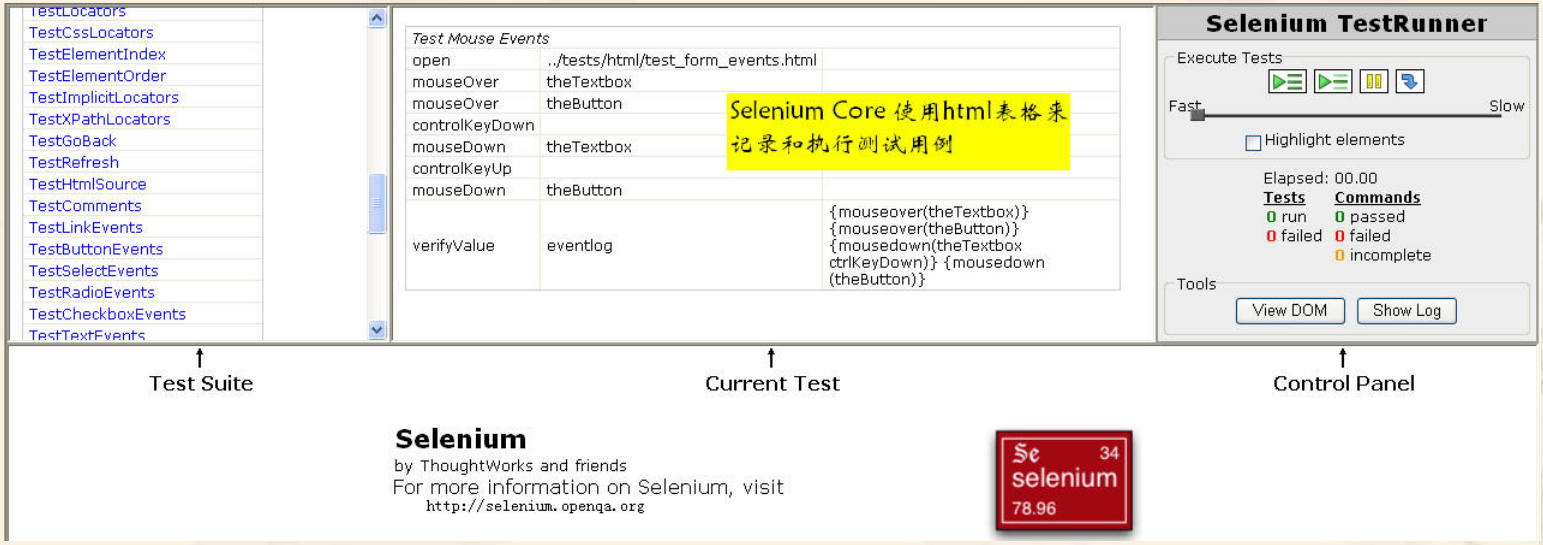
- 性能测试由于涉及面广，对测试人员的要求就很高。我想，性能测试人员应该培养如下几方面的能力：
  - 如前所述，对应用架构的透彻理解。
  - 沟通能力，测试进行过程中，一定要培养勤于跟开发沟通的意识，以提高工作效率。
  - 解决问题的能力，在编脚本或者测试执行过程中，会碰到很多问题。首先是不要害怕，先考虑问题的可能原因，然后一步步定位、验证。当然，这个过程，需要调试等经验的不断积累。

自动化测试原型

- 选用合适的工具模拟用户操作
  - Selenium
    - 跨平台、跨浏览器 、支持多种编程语言、支持Ajax测试的Web自动测试框架
    - Google内部使用范围最广的Web自动测试工具
      - 超过50个团队在使用
      - 平均每天要跑51000多个测试用例
    - 开源，所以不需要license。

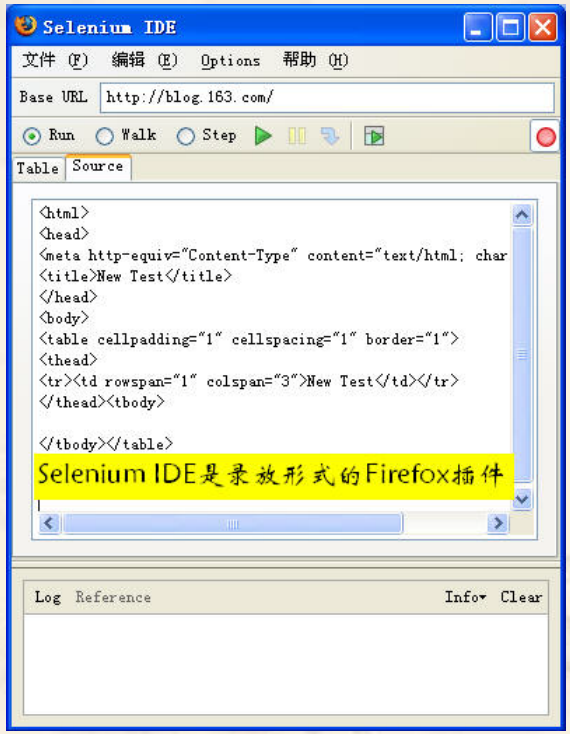
自动化测试原型

- Selenium Core(Jason Huggins)
  - 内核用js实现，开放API接口来模拟用户的鼠标键盘操作（如click、doubleclick 、 type 、close 、 wait等）
  - 运行于真实的浏览器上，如Firefox, IE, Opera等



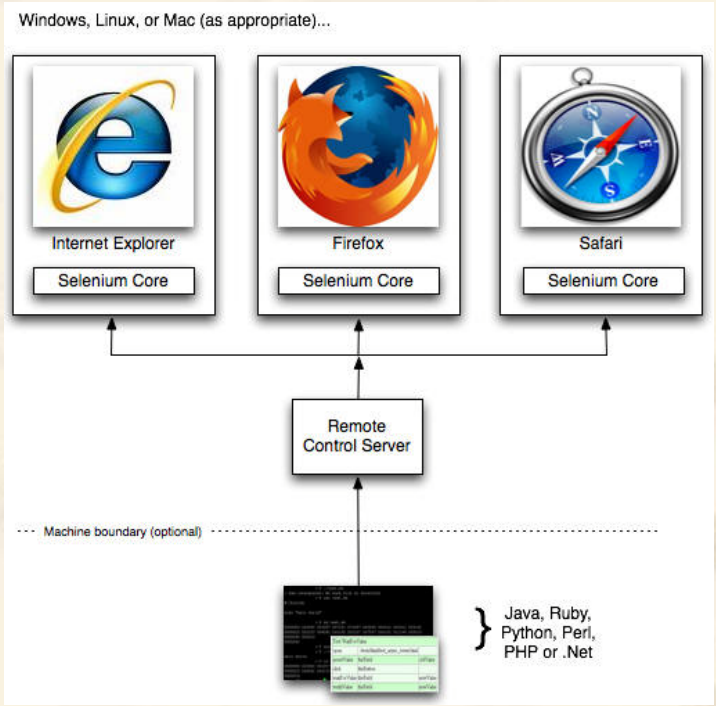
自动化测试原型

- Selenium IDE
  - Firefox插件
  - 轻松录制和回放用户操作
  - 将测试导出存储为HTML、Ruby 、JAVA脚本
  - 可再手工编辑



自动化测试原型

- SeleInium Remote Control(Paul Hammant)
  - 目前支持.NET、Java、Ruby、Perl、PHP、Python等编程语言
  - 可选择使用自己熟悉的编程语言对应的Client Driver来驱动Server端执行测试



问题一

- Selenium基本满足了Web自动化测试要求
- 但随着自动测试的继续深入，我们发现
  - 测试用例无法保证其准确度及稳定性，在不同环境条件（时间，网络条件，账号等）下，多次测试结果大相径庭
  - 测试中遇到各种外部异常情况无法自行恢复
  - 用例失败很难判断是否是真的BUG

自动化测试原型

- 准确并可重用地定位页面元素
  - 优先使用元素的ID，Name来定位
  - 构造xpath表达式来定位
    - //button[contains(@onclick,'')]
    - //span[text()='']
    - // input[contains(text(), '') and contains(@id,'')]

选取较稳定的属性，与页面样式等易变因素减少耦合!

自动化测试原型

- 精确模拟Ajax异步操作
  - 异步等待精确的加载条件方可下一步操作
  - 用WaitForCondition，而不用Thread.sleep(…)
- 判断与处理各种异常情况
  - 测试开始前进行数据清理，结束后执行异常恢复
  - 减轻用例之间的依赖与耦合关系

问题二

- 随着用例数量的不断增加，执行测试时需要组织与管理
- 数据驱动的测试要求运行时可以设置覆盖不同路径的测试数据
- 测试结果的查看不便，导致异常发现的时效性很低

搭建自动化测试框架

- 选用合适的测试框架组织管理用例
  - TestNG
    - 由JUnit演变而来更灵活易用的系统测试框架
    - 在测试代码中插入TestNG Annotations 标识
      - @Test @Parameters
      - @BeforeSuite @AfterSuite
      - @BeforeTest @AfterTest
    - 支持数据驱动测试：Parameter, DataProvider
    - 使用XML来描述测试集合suite与测试用例test

搭建自动化测试框架

- 构建测试集合xml的矩阵组织结构
  - 在TestNG基础上扩展XML组织结构
  - 增加产品product，模块module对用例进行组织
  - 另外每个用例还可从属与某服务或接口

搭建自动化测试框架

- 运行时可以按照产品，模块，用例，服务等不同维度从用例库supertest.xml中挑选合适的用例及测试帐号
- 自动生成TestNG运行需要的测试集合Xml
- 增加Listener实时监听测试用例失败、通过，测试集合开始、结束事件。

搭建自动化测试框架

- 测试结果的统计与分析
  - 在测试用例或配置失败时立即发邮件提醒，提供错误信息，当时使用的测试数据及失败截图
  - 在测试集合结束时统计总体失败率及各模块或服务的失败率，生成测试结果分析报告

测试用例框架的搭建

- 将单元操作进行组合封装
  - 基于selenium提供的单元操作API进行二次封装
  - 对AJAX系列操作进行组合
  - 提供动态拼接表达式的简化操作接口
- 封装重用性高的功能点
  - 设计底层公用的库函数，如写日志、发留言、上传相片等，开放调用

开发人员的配合

- 页面元素尽量使用ID，name来命名
- ID，name等重要属性尽可能保持稳定
- 将动态数据填充进页面隐藏Div，供测试时动态获取

```
<#if (seleniumTestOn?default("off") == "on")>
<div id="divSuffixEditBlogAll" style="display:none">
        <div id="sufBlog"></div>
        <div id="sufCom"></div>
</div>
</#if>
```

问题三

- 脚本启动方式对于普通测试人员来说，执行测试不够直观易用
- 测试开始后测试人员无法判断测试的实时进度

搭建自动化测试平台

- 自动测试平台（http://qa.163.org/test）
  - 各产品常规测试服务

搭建自动化测试平台

- 各产品常规测试：
  - 测试内容的选择支持同时挑选多项不同产品和模块，还可以细化到测试用例级别对于选中的测试内容进行筛选
  - 测试用例范围涵盖多项产品
  - 设计开发几百个测试用例并广泛应用到版本回归测试，验收测试，线上每日回归测试中

搭建自动化测试平台

- 测试人员可以通过浏览器远程访问平台，方便灵活地查看及选择各产品和模块对应的测试内容，选择或自定义测试环境，执行测试。
- 主控服务器接收到测试请求后，将不同测试环境的请求分派到不同测试服务器上。
- 测试服务器根据用户请求的参数挑选合适用例，从共享队列中获取测试端口和账号，自动生成测试集合xml并开始执行测试。

搭建自动化测试平台

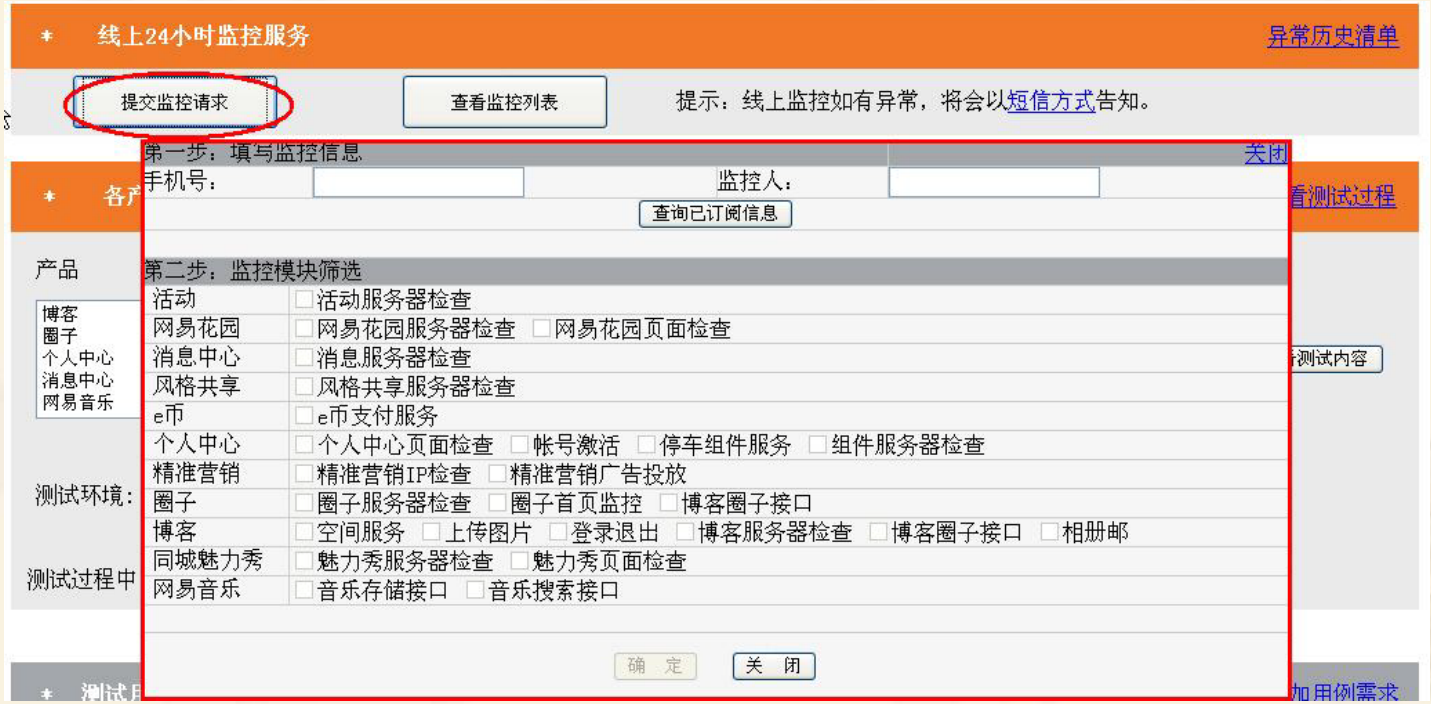
- 测试过程中测试服务器在共享文件夹建立本次请求进度跟踪文件，实时监听测试执行情况并更新进度文件，主控服务器在页面端间隔异步刷新来呈现进度。
- 测试完成后将测试报告远程拷贝给主控服务器，呈现在页面端。
- 如有测试用例失败，测试人员可直接运行失败用例，主控机将按照上面的流程调用测试服务器生成失败用例的xml重新执行。

问题四

- 随着监控需求与种类的增多，不同监控服务对于监控要求不一致且经常变化，监控管理混乱
- 短信报警准确率不够，偶尔一次失败并不能判定服务异常
- 报警提供的信息少，不易定位问题

搭建线上监控平台

- 自动测试平台（http://qa.163.org/test）
  - 线上24小时监控服务



搭建线上监控平台

- 监控发现异常后会以短信形式报警，内容包含出错的服务器名称及我们提供的失败原因分析
  - 报警时自动检查网络，数据库，应用服务器状态，附加在报警短信中
- 由单次失败报警改为按照服务类型累计失败次数，连续一定次数失败才予报警
  - 是否需要累计报警以及累计次数可根据需求对每个服务单独进行配置

搭建线上监控平台

- 测试服务器上早上6：00到凌晨01：00无间断连续运行监控用例，当某服务的失败率超过预设值时，读取订阅文件，给该服务相关订阅人发短信报警
- 监控频率按照优先级分为10分钟到1小时不等

自动化测试的说明

- 这方面，我们组有人比我更精通
- 以后如果有机会，可以由他们进一步交流经验

最后

- 感谢CCTV，感谢MTV。。
- 感谢大家
- 感谢并欣赏淘宝测试组，不吝投入人力和物力，来促进杭州测试业界的交流。。