**网银在线性能测试指南**

[1 序言 5](#_Toc377939382)

[1.1 编者寄语 5](#_Toc377939383)

[1.2 编者简介 5](#_Toc377939384)

[2 性能测试方法与实践 5](#_Toc377939385)

[2.1 性能测试目的 5](#_Toc377939386)

[2.2 性能测试类型 6](#_Toc377939387)

[2.3 性能测试基本过程 7](#_Toc377939388)

[2.4 性能评估模型 9](#_Toc377939389)

[2.4.1 性能评估模型概述 9](#_Toc377939390)

[2.4.2 闭环流程图 13](#_Toc377939391)

[2.4.3 TPS计算公式 14](#_Toc377939392)

[2.4.4 影响因子 14](#_Toc377939393)

[2.4.5 响应时间 15](#_Toc377939394)

[2.4.6 稳定性要求 15](#_Toc377939395)

[2.4.7 性能测试环境要求 16](#_Toc377939396)

[2.4.8 并发量与吞吐量（TPS）的关系 16](#_Toc377939397)

[2.4.9 线上性能数据采集 16](#_Toc377939398)

[2.5 性能测试需求分析 16](#_Toc377939399)

[2.5.1 业务调研 17](#_Toc377939400)

[2.5.2 性能需求评估 18](#_Toc377939401)

[2.5.3 性能测试指标 19](#_Toc377939402)

[2.5.3.1 性能指标分析 19](#_Toc377939403)

[2.5.3.2 性能指标详解 20](#_Toc377939404)

[2.5.3.3 性能指标参考 22](#_Toc377939405)

[2.5.4 压力与容量预估 23](#_Toc377939406)

[2.6 性能测试准备 23](#_Toc377939407)

[2.6.1 性能测试环境申请 23](#_Toc377939408)

[2.6.2 环境清理 24](#_Toc377939409)

[2.6.3 环境搭建及数据准备 24](#_Toc377939410)

[2.6.4 压力工具选择 24](#_Toc377939411)

[2.6.5 资源监控工具部署 26](#_Toc377939412)

[2.7 性能测试执行 26](#_Toc377939413)

[2.7.1 无人值守执行性能测试 26](#_Toc377939414)

[2.7.2 动态调优 27](#_Toc377939415)

[2.7.3 边执行边思考 27](#_Toc377939416)

[2.8 性能分析与调优 28](#_Toc377939417)

[2.9 性能测试报告与总结 28](#_Toc377939418)

[2.10 性能准入标准与建议 28](#_Toc377939419)

[3 性能分析与调优 30](#_Toc377939420)

[3.1 性能分析 30](#_Toc377939421)

[3.1.1 性能分析的目的 30](#_Toc377939422)

[3.1.2 常见性能瓶颈征兆 30](#_Toc377939423)

[3.1.3 性能数据解读建议 31](#_Toc377939424)

[3.1.4 如何定位性能问题 32](#_Toc377939425)

[3.1.5 常见性能问题参考 34](#_Toc377939426)

[3.2 性能调优 35](#_Toc377939427)

[3.2.1 性能调优目标与策略 35](#_Toc377939428)

[3.2.2 调优的原则 36](#_Toc377939429)

[3.2.3 调优的基本步骤 36](#_Toc377939430)

[3.2.4 性能调优的层次 36](#_Toc377939431)

[3.2.4.1 设计调优 36](#_Toc377939432)

[3.2.4.2 算法调优 37](#_Toc377939433)

[3.2.4.3 代码调优 38](#_Toc377939434)

[3.2.4.4 JVM调优 38](#_Toc377939435)

[3.2.4.5 网络调优 38](#_Toc377939436)

[3.2.4.6 系统调优 41](#_Toc377939437)

[3.2.4.7 数据库调优 43](#_Toc377939438)

[3.2.5 配置项性能调优参考 46](#_Toc377939439)

[4 性能测试模板 51](#_Toc377939440)

[4.1 COD系统性能测试报告范例 51](#_Toc377939441)

[4.2 性能测试报告模板 65](#_Toc377939449)

[5 性能测试工具 65](#_Toc377939450)

[5.1 JMeter参考手册 65](#_Toc377939451)

[5.1.1 JMeter介绍 65](#_Toc377939452)

[5.1.2 JMeter安装 66](#_Toc377939453)

[5.1.3 JMeter运行 67](#_Toc377939454)

[5.1.3.1 命令行模式运行 67](#_Toc377939455)

[5.1.3.2 集群模式运行 67](#_Toc377939456)

[5.1.3.3 命令行选项 67](#_Toc377939457)

[5.1.4 JMeter常用协议 68](#_Toc377939458)

[5.1.4.1 HTTP请求 68](#_Toc377939459)

[5.1.4.2 https请求 69](#_Toc377939460)

[5.1.4.3 Java请求 70](#_Toc377939461)

[5.1.5 JMeter常用场景 72](#_Toc377939462)

[5.1.5.1 稳定TPS测试 72](#_Toc377939463)

[5.1.5.2 并发压力测试——设置集合点 72](#_Toc377939464)

[5.1.5.3 混合压力场景测试 72](#_Toc377939465)

[5.1.5.4 参数化 73](#_Toc377939466)

[5.1.6 JMete主要组件介绍 75](#_Toc377939467)

[5.1.6.1 线程组 75](#_Toc377939468)

[5.1.6.2 控制器 76](#_Toc377939469)

[5.1.6.3 取样器 76](#_Toc377939470)

[5.1.6.4 逻辑控制器 77](#_Toc377939471)

[5.1.6.5 监听器 78](#_Toc377939472)

[5.1.6.6 定时器 78](#_Toc377939473)

[5.1.6.7 断言 79](#_Toc377939474)

[5.1.6.8 配置元件 79](#_Toc377939475)

[5.1.6.9 前置处理器 79](#_Toc377939476)

[5.1.6.10 后置处理器 80](#_Toc377939477)

[5.1.6.11 执行顺序 80](#_Toc377939478)

[5.1.7 JMeter函数 80](#_Toc377939479)

[5.1.7.1 函数列表 80](#_Toc377939480)

[5.1.8 JMeter扩展 81](#_Toc377939481)

[5.1.8.1 JMeter监控统计插件 81](#_Toc377939482)

[5.1.9 JMeter持续集成 84](#_Toc377939483)

[5.1.9.1 JMeter+Ant 84](#_Toc377939484)

[5.1.9.2 JMeter+Maven 85](#_Toc377939485)

[5.1.9.3 JMeter+Jenkins 86](#_Toc377939486)

[5.1.10 JMeter+Selenium 87](#_Toc377939487)

[5.1.10.1 使用JMeter JUnit Sampler 87](#_Toc377939488)

[5.1.10.2 使用BeanShell Sampler 90](#_Toc377939489)

[5.1.10.3 使用JMeter Plugins的WebDriver Sampler 91](#_Toc377939490)

[5.2 Loadrunner 91](#_Toc377939491)

[5.2.1 loadrunner简介 91](#_Toc377939492)

[5.2.2 录制及调试脚本 92](#_Toc377939493)

[5.2.2.1 录制脚本前的设置： 92](#_Toc377939494)

[5.2.2.2 Run-time Setting设置 94](#_Toc377939495)

[5.2.2.3 其它设置 98](#_Toc377939496)

[5.2.2.4 参数化 98](#_Toc377939497)

[5.2.2.5 检查点 101](#_Toc377939498)

[5.2.2.6 集合点 101](#_Toc377939499)

[5.2.2.7 事务 102](#_Toc377939500)

[5.2.2.8 关联 102](#_Toc377939501)

[5.2.2.9 思考时间 105](#_Toc377939502)

[5.2.2.10 JavaVuser协议例子 105](#_Toc377939503)

[5.2.3 场景设置 106](#_Toc377939504)

[5.2.3.1 如何设置并发用户数 106](#_Toc377939505)

[5.2.3.2 设置压力机 107](#_Toc377939506)

[5.2.3.3 监控系统资源情况 107](#_Toc377939507)

[5.2.3.4 集合点策略 109](#_Toc377939508)

[5.2.4 结果分析 109](#_Toc377939509)

[5.2.4.1 Analysis Summary 109](#_Toc377939510)

[5.2.4.2 Running Vuser 110](#_Toc377939511)

[5.2.4.3 Transaction Response Time(Percentile) 110](#_Toc377939512)

[5.2.4.4 合并图 111](#_Toc377939513)

[5.2.5 常见函数 111](#_Toc377939514)

[5.3 JVM监控工具 112](#_Toc377939515)

[5.3.1 Java自带的性能监控工具 112](#_Toc377939516)

[5.3.1.1 具体案例： 115](#_Toc377939517)

[5.3.2 Eclipse Mat Usage 116](#_Toc377939518)

[5.3.2.1 生成 Java heap Dump 文件 116](#_Toc377939519)

[5.3.2.2 MAT安装 116](#_Toc377939520)

[5.3.3 JProfiler 119](#_Toc377939521)

[5.3.3.1 JProfiler在linux上的配置使用 119](#_Toc377939522)

[5.4 OS与网络监控与分析工具 126](#_Toc377939523)

[5.4.1 工具总览 126](#_Toc377939524)

[5.4.2 top 126](#_Toc377939525)

[5.4.3 vmstat 127](#_Toc377939526)

[5.4.3.1 常用命令：vmstat 1 128](#_Toc377939527)

[5.4.4 iostat 128](#_Toc377939528)

[5.4.4.1 常用命令：iostat -x 129](#_Toc377939529)

[5.4.5 netstat 129](#_Toc377939530)

[5.4.5.1 常用命令：netstat -anltp 130](#_Toc377939531)

[5.4.6 dstat 130](#_Toc377939532)

[5.4.6.1 常用命令：dstat -cmsdnlpyt 130](#_Toc377939533)

[5.4.7 Nmon 130](#_Toc377939534)

[5.4.8 文件句柄数查看及修改 133](#_Toc377939535)

[5.5 数据库分析与诊断工具 134](#_Toc377939536)

[5.5.1 Oracle AWR 134](#_Toc377939537)

[5.5.2 MySQL AWR 144](#_Toc377939538)

[5.5.2.1 What is myawr 144](#_Toc377939539)

[5.5.2.2 Myawr Data Model 146](#_Toc377939540)

[5.5.2.3 Quickstart 147](#_Toc377939541)

[5.5.2.4 Dependencies 149](#_Toc377939542)

[5.5.2.5 Mysql WorkLoad Report 150](#_Toc377939543)

[5.5.2.6 Mysql Snapshot Report 152](#_Toc377939544)

[5.5.2.7 MySQL AWR 部署和报告生成 153](#_Toc377939545)

# 序言

## 编者寄语

网银在线做为一家支付公司，在加入京东后，除了对功能的要求很高外，对性能的要求也越来越高。所对应的性能测试范围也越来越广，包括底层框架、web应用、接入api、以及后续的无线等。因此质量部成立了性能测试topic及其虚拟团队，负责研究性能测试的相关理论与实践，并在此基础上产出了《性能测试指南》

性能测试指南的目的在于授人以渔而不是授人以鱼，通过在指南的指引下，完成性能测试，并掌握性能测试过程中所需要的工具、技能、方法等。积累相关的性能测试经验，并提升测试人员自身的技能。

性能测试指南主要分为4个部分：方法与实践篇，性能分析与调优篇，性能测试模板，性能测试工具篇。其中方法与实践主要讲述性能测试的过程，包含性能测试过程，性能评估模型，需求分析方法，常用性能测试指标等。性能分析与调优讲解了常用的分析定位方法以及性能调优的一些方法与层次。性能测试模板则提供了一个性能测试报告范例与性能测试模板。工具篇讲述了常用的工具集及使用方法。

每次进行性能测试以及发现性能瓶颈的过程都是一次经验的累积与学习的过程，在解决问题的时候我们会学到很多的技能和方法，希望性能测试指南能为大家提供一个方向的指引。

## 编者简介

|  |  |
| --- | --- |
| 作者 | 编者职责 |
| 史汉青 | 性能测试指南的倡导者与发起人，提出编写性能测试指南这一战略思想。对指南的架构提出了一些建设性的意见，并对文档内容进行把关。保证了文档的严谨性。 |
| 韩钰 | 性能测试指南编辑员，负责性能评估模型，需求分析，测试准备，测试执行等章节的编写。 |
| 杨飞菲 | 性能测试指南编辑员，负责jmeter参考手册，os与网络监控分析工具，工具对比等章节的编写。 |
| 欧明宝 | 性能测试指南编辑员，负责提供excel性能测试报告模板及范例，并负责jvm监控工具，数据库分析与诊断工具，配置项性能调优参考等章节的编写 |
| 李俊亭 | 性能测试指南编辑员，负责loadrunner工具章节的编写，以及提供word版本测试报告模板与范例。 |
| 乔健 | 性能测试指南编辑员，负责性能准入标准与建议章节的编写。 |
| 袁巨星 | 性能测试指南编辑员，负责指南文档架构，以及测试基本概念（目的、类型、基本过程、需求分析方法），性能测试指标，性能分析与调优等章节的编写。 |

# 性能测试方法与实践

本章节做为《性能测试指南》的核心章节，主要阐述性能测试的过程，以及该过程中所涉及到的一些方法论。具体包括性能评估模型，性能需求分析，性能准备，性能执行等。

## 性能测试目的

性能测试的最终目标是为了最大限度的满足用户的需求，我们通常为了达到以下目标而进行性能测试：

(1)评估系统的能力：测试中得到的负荷和响应时间数据可以被用于验证所计划的模型的能力，并帮助作出决策；

(2)寻找系统瓶颈，进行系统调优；

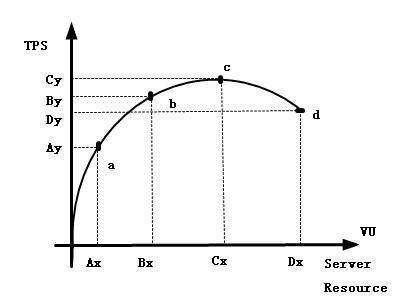
(4)检测软件中的问题；

(5)验证稳定性、可靠性；

## 性能测试类型

在讲述性能测试类型前，先简单分析性能测试的压力模型。

如下图所示，随着单位时间流量的不断增长，被测系统的压力不断增大，服务器资源会不断被消耗，TPS值会因为这些因素而发生变化，而且符合通常情况下的规律。

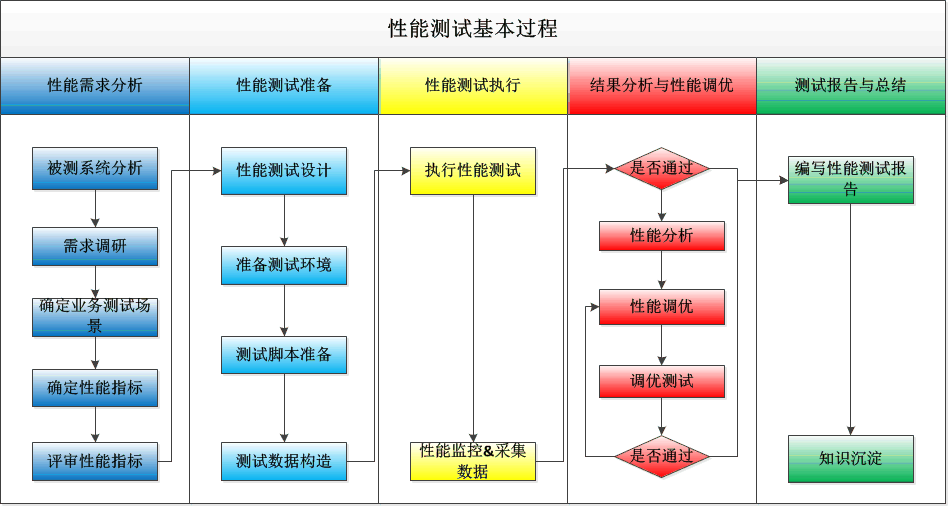


说明：  
a点：性能期望值  
b点：高于期望，系统资源处于临界点  
c点：高于期望，性能处于拐点  
d点：超过负载，资源不够用，系统处于崩溃  
通过如上模型图中的情况，我们大致可以将当前性能测试分成如下4类：性能测试、负载测试、压力测试、稳定性测试，具体的特性及描述，请参考下表：

| 测试类型 | 简述 | 测试目的和特点 | 关注点及产出 |
| --- | --- | --- | --- |
| 性能测试 | 性能测试是指通过模拟生产运行的业务压力量和使用场景组合，测试系统的性能是否满足生产性能要求。 | 目的：验证系统是否有其宣称具有的能力。  特点：对系统性能已经有了解的前提，对需求有明确的目标，并在已经确定的环境下进行的。 | 关注的是系统性能是否和具体的性能需求相一致，而当系统性能超过性能需求的时候，系统的表现并不是测试人员关心的重点。 |
| 负载测试 | 是指对系统不断地增加压力或增加一定压力下的持续时间，直到系统的某项或多项性能指标达到安全临界值，例如某种资源已经达到饱和状态等 | 目的: 找到系统处理能力的极限。了解系统的性能容量，或是配合性能调优来使用。 | 1.得出线下系统最有TPS。  2.得出线下系统最有TPS时系统资源利用率。  3.得出线下系统极限并发数。 |
| 压力测试 | 压力测试是评估系统处于或超过预期负载时系统的运行情况。压力测试的关注点在于系统在峰值负载或超出最大载荷情况下的处理能力。 | 目的：检查系统处于大压力性能下时，应用的表现。  特点：一般通过模拟负载等方法，使得系统的资源使用达到较高的水平。 | 关注点:发现功能测试不能发现的非功能性缺陷。  产出：  1.得出线下系统崩溃点的TPS。  2.得出线下系统崩溃时资源使用率  3.得出线下系统极限并发数 |
| 稳定性测试 | 在给系统加载一定业务压力的情况下，使系统运行一段时间，以此检测系统是否稳定。 | 目的：主要目的是验证是否支持长期稳定的运行。 | 关注系统稳定性。  得出系统稳定状态下的资源利用、连接池、TPS、响应时间、DB健康情况等数据。 |

## [性能测试基本过程](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1968324)

性能测试从实际执行层面来看，测试的过程一般分为这么几个阶段，见图：



1.性能需求分析

性能需求分析是整个性能测试工作开展的基础，如果连性能的需求都没弄清楚，后面的性能测试工具以及执行就无从谈起了。

在这一阶段，性能测试人员需要与PM、DEV及项目相关的人员进行沟通，同时收集各种项目资料，对系统进行分析，确认测试的目标。并将其转化为可衡量的具体性能指标。

测试需求分析阶段的主要任务是分析被测系统及其性能需求，建立性能测试数据模型，分析性能需求，确定合理性能目标，并进行评审；

2.性能测试准备

主要包括：设计场景，根据场景编写程序、编写脚本、准备测试环境，构造测试数据，环境预调优等；

针对系统的特点设计出合理的测试场景。为了让测试结果更加准确，这里需要很细致的工作。如建立用户模型，只有知道真实的用户是如何对系统产生压力，才可以设计出有代表性的压力测试场景。这就涉及到很多信息，如用户群的分布、各类型用户用到的功能、用户的使用习惯、工作时间段、系统各模块压力分布等等。只有从多方面不断的积累这种数据，才会让压力场景更有意义。最后将设计场景转换成具体的用例。

测试数据的设计也是一个重点且容易出问题的地方。生成测试数据量达到未来预期数量只是最基础的一步，更需要考虑的是数据的分布是否合理，需要仔细的确认程序中使用到的各种查询条件，这些重点列的数值要尽可能的模拟真实的数据分布， 否则测试的结果可能是无效的。

预调优指根据系统的特点和团队的经验，提前对系统的各个方面做一些优化调整，避免测试执行过程中的无谓返工。比如一个高并发的系统，10000人在线，连接池和线程池的配置还用默认的，显然是会测出问题的。

3.执行性能测试

执行阶段工作主要包含两个方面的内容：一是执行测试用例模型，包括执行脚本和场景；其次测试过程监控，包括测试结果、记录性能指标和性能计数器的值

4.结果分析与性能调优

发现问题或者性能指标达不到预期，及时的分析定位，处理后重复测试过程。性能问题通常是相互关联相互影响的，表面上看到的现象很可能不是根本问题，而是另一处出现问题后引起的反应。这就要求监控收集数据时要全面，从多方面多个角度去判断定位。调优的过程其实也是一种平衡的过程，在系统的多个方面达到一个平衡即可。

5.性能报告与总结

编写性能测试报告，阐明性能测试目标、性能结果、测试环境、数据构造规则、遇到的问题和解决办法等。并对此次性能测试经验进行总结与沉淀。具体性能测试报告的编写可以参考《性能测试报告模板》。

上面所有内容中，如果排除技术上的问题，性能测试中最难做好的，就是用户模型的分析。它直接决定了压力测试场景是否能够有效的模拟真实世界压力，而正是这种对真实压力的模拟，才使性能测试有了更大的意义。可以说，性能测试做到一定程度，差距就体现在了模型建立上。

　　至于性能问题的分析、定位或者调优，很大程度是一种技术问题，需要多方面的专业知识。数据库、操作系统、网络、开发都是一个合格的性能测试人员需要拥有的技能，只有这样，才能从多角度全方位的去考虑分析问题。

## 性能评估模型

### 性能评估模型概述

**我们的系统性能到底能不能够支撑线上真实大量的订单交易？**

我想，这是我们每一个网银在线的同学都很关心的问题，也是性能评估模型篇需要解答的最终问题。所以我们就带着这个问题来一步步深入性能测试。

本问题的难度不在于一个简单的结果，而在于答案背后的一系列性能测试的评估数据和算法，以及如何建立一个良好可持续的“网银在线性能评估模型”。

通常来讲，性能测试是指通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。

而要回答“能否支撑线上真实的订单交易”这样带有预测性的问题，实际上还需要用上另一种手段，即“**性能预测**”，而“网银在线性能评估模型”就是用来做性能预测的。

**在预测之前，我们先来做一个数据分析，通过这个分析我们可以大概了解线上与线下的推算过程。**

2013年11月11日，支付宝实现了当天交易总金额**350亿元**，订单总数**1.8亿笔**（其中手机支付占24%），活跃用户**1.2亿**。（来源：支付宝官方微博<http://weibo.com/1627897870/AiiAjEwHO>）

显然这是一个非常震惊的数字，它见证着电商的今天也预示着电商的未来。

**假设有一天，网银在线也同样需要面对如此巨大的成交量**，当然这会是我们的骄傲。可问题也会随之而来，我们的系统能不能撑得住？我们有多少个性能瓶颈？我们应该给哪些服务加机器？我们的性能测试应该如何做？我们应该设定哪些性能指标？稳定性99%够不够？等等。希望在这一天到来之前我们就已经很好的解决了上面提到的问题。针对这个数字，下面我们就一起来剖析数字背后的性能情况。

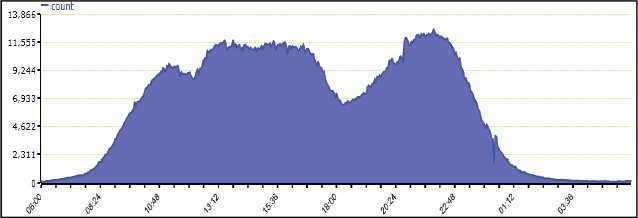
双11当天，支付宝的订单数是1.8亿笔，意味着每小时订单数达到1.8亿 / 24 = 750万笔，也意味着每秒订单数达到750万 / 3600 = 2083笔。

首先，让我们先来看看**吞吐率指标（TPS）**，假设支付宝有100台前置服务器，分到每台机器就是每秒处理订单2083笔 / 100 = 20笔。

这是不是意味着只要单台服务器的性能达到20 tps，我们的线上服务就足以支撑每天1.8亿的交易量呢？

答案显然不是。

我们先来看看淘宝网每天交易量分布图：

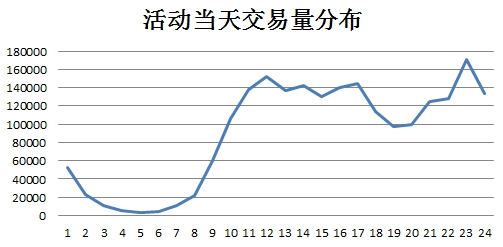


淘宝网日常每天交易量分布

可以看到，交易量并不是24小时平均分布的，从早上8点开始到晚上12点才是交易量发生的主要时间段，也就是说实际上每天只有2/3的时间（16小时）才是有效时间。我们的性能指标应该调整到20tps \* 150% = 30tps。

即使是16个小时中，交易量也不是平均的，在午饭和晚饭时间会出现两个波谷，晚上8点到11点之间是一天的最高峰。最高峰大约是平均值的1.2倍，如果我们不想损失掉高峰时期的交易量的话，我们的吞吐率指标需要再上调到30tps \* 1.2 = 36tps。

以上只是一个普通日子的交易量分布情况，如果有某些重大活动事件，比如限时抢购、定时秒杀等活动，那么我们很可能在短时间承受几倍甚至十几倍的压力。如下图是京东某年某活动当天的交易量分布图：



京东某活动当天交易量分布图

总体交易量是平日的几倍到十几倍，交易量从零点开始就达到一个小高峰，凌晨2点后逐渐降到冰点，早上8点又开始了一天的持续热度，下午7点出现一个小波谷紧接着就迎来了晚上11点左右的最高峰。对于这种特别庞大的活动我们需要提前做好性能预案，以往的数据就是我们最好的参考。

通常来讲，我们对于系统的性能要求在每天高峰时期的1.5倍到4倍，如果按2倍计算，那么我们的吞吐率指标就继续上调到 36tps \* 2 = 72tps。

好了，单台服务器的性能达标了72tps。但是100台服务器加起来就能承载 72tps \* 100 = 7200tps吗？

比较难，这依赖于我们的**负载均衡**算法。假设我们有一个还不错的算法，但也不能保证长时间绝对的公平，所以单台服务器还需要做好压力波动的缓冲，如果按1.5倍计算，这样我们的指标又要上调到72tps\*150% = 98tps。

这里，我们已经把对单台服务器的性能指标从20tps逐渐上调到近100tps，是原来的5倍。当然这里还远没有结束。这1.8亿笔只是单一的下单操作的PV，假设用户每下单一笔，需要伴随着10次查询，3次提交，那么我们的服务器需要再评估其他或查询或提交的性能。假设这些其他附带请求的压力总和可能还得增大5倍，即达到98tps \* 5 = 500tps。

还没完，这只是100台前置服务器，后面还有我们的核心业务层、消息系统、数据库系统、第三方依赖等等。它们中的任何一方都有可能成为性能瓶颈，虽然我们的分析假设了对外依赖都不是瓶颈的情况，但实际上往往就是某些不起眼的有限资源的依赖导致了整体吞吐率的下降。

这是一个水桶效应，装水量是由最小的那块板决定的。显然我们有必要对每一个子系统（公司内和公司外）分别做单独的性能测试和评估。

完了吗？可能还没有。假设我们的稳定性足够好，好到99.99%，即使这样，在面对超大量订单（1.8亿）的时候也还是意味着有1.8万个订单是失败的。

每一次失败对用户来讲都是一次糟糕的体验，可能还会因此引起一些连锁反应。所以稳定性也是系统性能的重要指标，我们做性能测试不能仅仅关注数量，还要关注质量。

对了，既然我们已经提到用户体验，那就不得不谈**响应时间（RT）**了，这是性能指标的又一个维度。

一般用户不会关心你有多少台服务器，也不会关心服务器能支撑多大的交易量，他们只关心你让他等了多长的时间。关于这方面的研究有不少，较新的观点可能是：



用户的耐心似乎一直在下降。不管怎么样，从性能角度上看，用户的诉求就是一句话：“不要让我等！”。

为了不让用户等，我们的性能测试需要针对不同场景给出不同的考察指标，比如前端页面加载时间，后端系统响应时间，并统计出最大、最小、平均值、标准差等等。

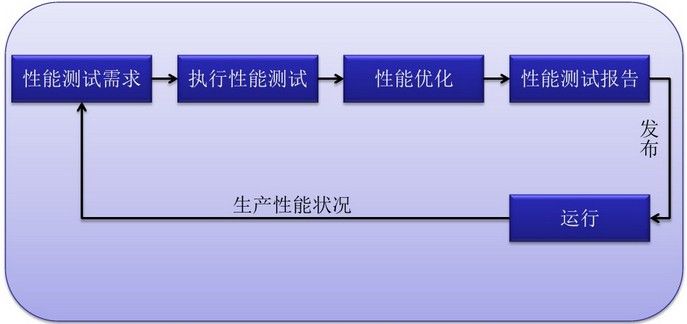
好吧，到这里我们已经层层分析，从TPS开始到用户体验上。需要注意的是，在分析中我们使用了一些模糊的参数，比如前面说1.5倍到4倍，那到底应该是多少呢？

我想不同的业务、不同的架构、甚至不同的工程师，都有可能导致不同的结果。

而对于性能预测来讲，最重要的是建立起一个系统良好的可持续的“网银在线性能评估模型”，将线上和线下性能跟踪形成闭环，并不断调整动态发展，那样我们就可以适应任何的差异化，就算某一次我们评估错了，我们可以及时调整纠正，下一次我们就能更准了。

### 闭环流程图

一个可持续的良好模型需要测试流程上的配合，下图是一个将线下性能测试与线上性能状况相连接的闭环流程图。



性能闭环流程图

与传统流程不同点在于：

* 性能测试的终点不再是发布上线，上线之后我们仍然会继续跟踪性能情况，并将结果用于下一次性能测试需求分析。
* 将线下与线上真正关联起来，并通过换算系数实现性能预测，系数可动态调整的。
* 全程性能跟踪，在生产环境中也加入性能监控，更快的解决线上性能问题。

### TPS计算公式

* **线下单机基线TPS =线上日均交易量 / 86400 / 机器数 \* 160% \* F**

注：86400为一天的秒数。

注：假设硬件软件环境条件大致相同。

注：F为影响因子，默认为1 。

关键系数160%的由来，简单点讲就是在每天约50%的时间里完成了80%的工作，80% / 50% = 160%。

该公式还只是最简单场景的抽象，真实的情况可能会很复杂（详见前面的数据示例分析），不过作为线下基线TPS来讲具备很重要的参考意义。

线下基线TPS代表着为了达到线上的交易量，我们线下至少应该满足的性能指标。我们在做性能测试和评估的时候，基线TPS是一定要体现并首先要达到的目标。

影响因子F是一个经验值，也是需要不断调优的，线上与线下的软硬件环境不同都会导致F值的变化，甚至不同的业务线可以有不同的值。当我们的性能沙箱建立、性能闭环成型之后，会更加稳定下来。

### 影响因子

我们的计算公式是建立在假设硬件软件环境条件大致相同的基础上。但是在现实中，线上生产环境与线下测试环境总会难以避免的出现不一样的情况，当出现不一致的情况后我们如何来估算？这里我们很难给出准确的推算公式，但是我们列出了可能的影响因素：

|  |  |  |
| --- | --- | --- |
| **影响因子** | **吞吐率（TPS）** | **响应时间（RT）** |
| **硬件环境** | | |
| **CPU核数越多** | 越高 | 可能越低 |
| **CPU频率越高** | 可能越高 | 越低 |
| **CPU支持64位指令** | 越高 | 可能越低 |
| **内存越大** | 可能越高 | 可能越低 |
| **磁盘空间越大** | 不明显 | 不明显 |
| **磁盘速率越高** | 可能越高 | 可能越低 |
| **网卡速率越高** | 越高 | 越低 |
| **服务器品牌** | 未知 | 未知 |
| **显卡性能越高** | 无影响 | 无影响 |
| **软件环境** | | |
| **操作系统** | 首选Linux | 首选Linux |
| **Linux版本越高** | 不一定（2.6以上） | 不一定（2.6以上） |
| **JDK版本越高** | 不明显（1.6） | 不明显（1.6） |
| **SWAP分区越高** | 不一定 | 不一定 |
| **与其他服务共用** | 具体分析 | 具体分析 |
| **业务特点（计算型、IO型）** | 具体分析 | 具体分析 |

### 响应时间

* **后端核心服务响应时间不超过100ms**
* **后端一般服务响应时间不超过300ms**
* **前端用户页面加载时间不超过3s**

注：这里是指平均响应时间。但性能测试报告必须给出最大、最小、平均、方差等数值。

### 稳定性要求

* **确保零崩溃， 99.9% 的稳定性。**

零崩溃是指在任何非外部原因的情况下，不管运行多长时间，都不能出现程序崩溃的情况。

99.9%的稳定性，意味着每千条交易中只能容许一条出错，也意味着任何可能导致稳定性不达标的改进都需要慎重考虑。比如我们引入一种新的技术能使TPS增大50%，但是稳定性下降到97%，那么这可能也不能接受。

### 性能测试环境要求

为了更准确的预测性能，我们必须尽可能的保证性能测试环境的稳定。

* **性能测试环境要求真实的物理机，一般不能是虚拟机。**
* **压力工具与被测环境不能在同一台机器部署。**
* **尽可能的利用晚上无人操作的时间来做性能测试。**

测试环境准备是一个很重要又很繁琐的工作，它的效率提升依赖于我们的环境搭建平台的建设。

### 并发量与吞吐量（TPS）的关系

并发量（或叫并发用户数）：是指同一时间点对业务功能同时操作的用户数。

吞吐量（TPS）：一段时间内系统处理用户的请求数量。

虽然两者关系并不紧密，但是往往会有同学将它们搞混淆。其实我们可以用多线程模型来解释他们，并发量就相当于线程数，吞吐量就是所有线程处理请求数之和。

通常我们做性能测试主要考察和评估的是TPS，而并发量是一个相对较虚的概念，比如我们说一个系统支持同时1k人操作，一般不在评估的范围内。

### 线上性能数据采集

为了闭环需要，线上抽样一台机器，**采集每天的PV数据，每隔5分钟采集一次，每天就有60/5 \* 24 = 288个点**。

每个点需要采集Linux资源、JVM、IO以及PV数据，这些数据将成为将来性能评估的重要依据。

平台化也是趋势，不过目前还不确定怎么做。

**定期消除异常数据噪音，以月为单位可观察规律变化。**

## 性能测试需求分析

对于性能测试，一般来说工具的使用不是大的问题，问题是前期的需求分析，要不要做性能测试？性能测试的目的是什么？要做或不做性能测试的理由是什么？如何来做性能测试?这些问题的答案都是通过分析性能测试需求得到的。以下将阐述性能需求分析的一些基本方法。

### 业务调研

为了合理评估与制定系统的性能指标，业务调研是一个必备的过程，业务调研主要包含以下几个方面的内容：

1. 系统信息调研

* 系统类型：系统的基本特性，如交易处理型系统、数据处理型系统等
* 架构部署：系统的整体架构、服务器部署方式
* 技术信息：系统运行平台、数据库产品、使用的中间件、协议及通讯方式等
* 业务信息：支持的业务类型、业务范围与功能、与其它系统的业务关系等
* 系统历史运行情况：目标TPS，用户数、PV等数据
* 系统数据规模：将来系统使用规模，历史系统数据规模

1. 业务信息调研

* 基本业务功能：系统的基本业务概念以及系统的业务种类与具体功能
* 关键业务逻辑处理流程：关键业务的业务流程、交易路径、交易数据、交易流程与时序图
* 交易列表：调查业务系统全部交易清单，了解交易的组合关系、执行顺序等
* 交易量信息：在不同时间粒度下统计单个交易处理量以及总交易量信息
* 业务目标/业务拓展计划：目前的生产业务量和用户数以及系统预期业务目标和本次测试预期业务指标

1. 文档资料调研

* 功能规格说明书
* 系统设计文档
* 生产运营统计
* 前期系统测试资料

业务调研涉及到的角色有以下几个：业务人员、开发人员、客户(如京东)、产品人员、运维人员、DBA等。

通过业务调研至少可以得到以下几个方面的产出：

* 项目背景
* 系统架构说明
* 系统拓扑说明
* 测试范围说明
* 交易路径描述
* 需要测试的特性
* 不需要测试的特性

### 性能需求评估

在实施性能测试之前，需要对被测项目做相应的评估。主要目的是明确是否需要做性能测试和确立性能点，明确该测什么、期望值是多少。测试期望值也会根据情况评估，要求被测系统能满足将来一定时间段的压力。

判断是否进行性能测试可以从以下几个方面进行思考：

a、从业务角度来分析，如果一个项目上去后使用的人数比较多，量比较大，就有做性能测试的必要，反之，如果一个项目上线后，没有几个人在用，无论系统多大，设计如何复杂，并发性的性能测试是没有必要做的，前期可以否决。

b、从系统架构角度来分析，如果一个系统采用的框架是老的系统框架，只是在此框架上增加一些应用，其实是没有必要做性能测试。如果一个系统采用的是一种新的框架，可以考虑做负载测试。

c、从实时性角度来分析，如果一个项目要求某个功能的响应时间，这个有作并发测试的可能性，在大并发量的场景下，查看这个功能的响应时间。

d、从数据库角度分析，很多情况下，性能测试是大数据量的并发访问、修改数据库，而瓶颈在于连接数据库池的数量，而非数据库本身的负载、吞吐能力。这时，可以结合DBA的建议，来决定是否来做性能测试。

如果要进行性能测试，接下来我们就需要确定相应的性能点。主要从以下 4 个维度进行确定：

1. 关键业务。

首要维度，是确定被测项目是否属于关键业务，有哪些主要的业务逻辑点，特别是跟交易相关的功能点。例如快捷签约、交易等接口。如果项目（或功能点）不属于关键业务（或关键业务点），则可转入第二、三、四个维度。

1. 日请求量。

第二个维度，是界定被测项目各功能点的日请求量。如果日请求量很高，系统压力很大，而且又是关键业务，该项目需要做性能测试；而且其关键业务点，可以被确定为性能点。

1. 逻辑复杂度。

第三个维度，是判定被测项目各功能点的逻辑复杂度。如果一个主要业务的日请求量不高，但是逻辑很复杂，则也需要通过性能测试。原因是，在分布式方式的调用中，当某一个环节响应较慢，就会影响到其它环节，造成雪崩效应。

1. 运营推广计划。

第四个维度，是根据运营的推广计划来判定待测系统未来的压力。未雨绸缪、防患于未然、降低运营风险是性能测试的主要目标。被测系统的性能不仅能满足当前压力，更需要满足未来一定时间段内的压力。因此，事先了解运营推广计划，对性能点的制定有很大的作用。

例如，运营计划做活动，要求系统每天能支撑多少 PV、多少 UV，或者一个季度后，需要能支撑多大的访问量等等数据。

当新项目（或功能点）属于运营重点推广计划范畴之内，则该项目（或功能点）也需要做性能测试。

5.其它

以上 4 个评估维护，是相辅相成、环环相扣的，它们合成一个维度集。在实际工作中应该具体问题具体分析。例如，当一个功能点不满足以上 4 个维度，但又属于内存高消耗、CPU高消耗时，也可列入性能测试点行列。

### 性能测试指标

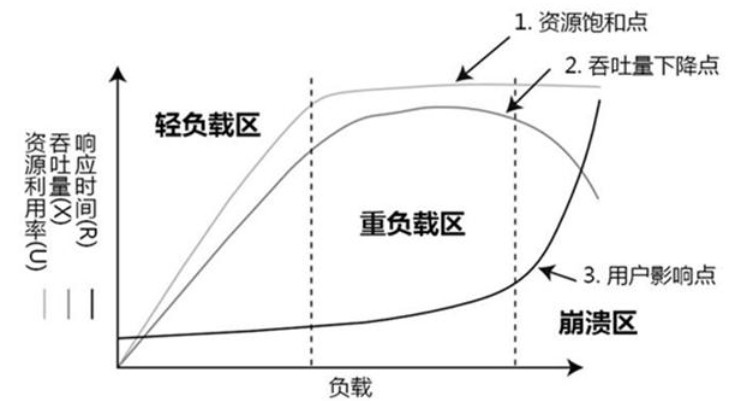
性能需求分析一个很重要的目标就是需要确定后期性能分析用的性能指标，性能指标有很多，可以根据具体项目选取和设定，而具体的指标值则需要根据业务特点和上述的一些方法进行设定。性能评估模型章节中也给了一些指标换算的基本方法。

#### 性能指标分析

判断一个系统的性能通常会取决于这三个最重要的性能指标：

* **吞吐率（TPS）**
* **响应时间（RT）**
* **系统资源利用率（Load）**

通常经验告诉我们，它们三者的趋势会如下图所示：



当处于轻负载区的时候，压力有多大，吞吐率就有多大。

一旦进入重负载区，就算是再怎么努力可能也会是白费。

如果到了崩溃区，那我们的努力就开始有反效果。

同样，对于我们做性能测试来讲，我们的目的就是找出这三个区域的分界线来。

所以在设计上，我们需要不断地递增负载来观察和评估系统性能情况。

比如下表（以下数据为伪造）：

|  |  |  |
| --- | --- | --- |
| **TPS** | **RT(AVG/MIN/MAX)** | **Load(cpu idle/mem/io)** |
| **100** | 50/15/80 | 93/8/2 |
| **200** | 50/18/90 | 91/8/2 |
| **400** | 55/18/145 | 87/9/4 |
| **800** | 130/25/560 | 70/10/14 |
| **1500** | 250/58/4488 | 48/11/75 |

可以看出，TPS低于400的时候基本属于轻负载区，400到800之间属于重负载区，高于800基本上属于崩溃区了。

#### 性能指标详解

该章节详细介绍性能测试过程中需要关注的性能指标，以及性能指标的范围。本指南中的所有的性能指标数据均可用指南中的监控工具进行收集。详细方法见监控工具章节。

##### 业务性能指标

|  |  |  |
| --- | --- | --- |
| **指标名** | **指标说明** | **采集方法** |
| 并发用户数 | 在同一时刻与服务器进行了交互的在线用户数量 | Jmeter/loadrunner |
| 响应时间 | 客户发出请求到得到响应的整个过程的时间。一般他可以分为三部分：呈现时间，数据传输时间和系统处理时间 | Jmeter/loadrunner |
| 吞吐量 | 一次性能测试过程中网络上传输的数据量的总和 | Jmeter/loadrunner |
| 每秒处理事务数(TPS) | 每秒钟系统能够处理事务或交易的数量，它是衡量系统处理能力的重要指标。 | Jmeter/loadrunner |
| 点击率 | 点击率可以看做是TPS的一种特定情况。每秒钟用户向web服务器提交的HTTP请求数。 | Jmeter/loadrunner |
| 事务成功率 | 本次测试中出现成功的事务数量/事务的总数 | Jmeter/loadrunner |

##### 应用服务器及硬件性能指标

|  |  |  |
| --- | --- | --- |
| **指标名** | **指标说明** | **采集方法** |
| CPU占用率 | 对一个时间段内CPU使用状况的统计。  建议：<75% | nmon/vmstat/top |
| Load Average | 一段时间内CPU正在处理以及等待CPU处理的进程数之和的统计信息，也就是CPU使用队列的长度的统计信息。  建议：<0.7\*CPU个数\*核数 | top/uptime |
| Paging rate | 内存页交换率，建议<80% | nmon/vmstat/top |
| 磁盘I/O | Iowait<30% | nmon/iostat/sar |
| SWAP | 有没有交换页面 | nmon |
| Tomcat启动的总线程数 | 关注线程数会不会无限制增长、线程数量是否足够 | Top |
| Full GC次数 | 关注Full GC次数，以及Full GC后内存占用有没有明显增长 | Jstat/jconsole |
| JVM内存使用与回收 |  | Jstat/jconsole |
| JDBC监控 |  | Jprofiler |
| Exception日志监控 | 监控压力过程中是否有异常日志产生 | Tail&grep |

##### 数据库性能指标

| **指标名** | **指标说明** | **采集方法** |
| --- | --- | --- |
| DB cpu占用率 | 数据库cpu利用率，建议<70% | nmon |
| DB load | 数据库服务器负载 | Top |
| DB mem | 内存使用是否平稳 | Nmon |
| DB 磁盘与I/O | I/O是数据库性能一个非常重要的因素，建议IOwait<30% | Nmon/iostat/sar |
| 数据库线程数 |  | Top |
| 缓存命中率 | 点击率可以看做是TPS的一种特定情况。每秒钟用户向web服务器提交的HTTP请求数。 | AWR |
| 共享池命中率 | 假如这个值低于95%就要考虑调整应用（改写多为变量绑定）或者增加内存 | AWR |
| Top 耗时 sql | 找出性能较差的sql，进行优化 | AWR |

#### 性能指标参考

在确定性能指标的时候，可以参考一下表中对应的数值。

|  |  |  |  |
| --- | --- | --- | --- |
| 指标项 | 优秀 | 中等 | 差 |
| TPS | TPS >=100 | 50<TPS <100 | TPS<=50 |
| 响应时间 | RT<=200ms | 200ms<RT<500ms | RT>=500ms |
| 响应长度 | ResponseSize<=100KB | 100KB<ResponseSize<500KB | ResponseSize>=500KB |
| CPU | CPU<=75% | 75%<CPU<80% | CPU>=80% |
| Load | Load<=2 | 2<Load<5 | Load>=5 |
| FullGC | FullGC<=1 | 1<FullGC<10 | FullGC>=10 |
| DBQPS | DBQPS<5\*TPS | 5\*TPS<DBQPS<10\*TPS | DBQPS>=10\*TPS |
| 慢SQL数量 | 慢SQL=0 | 慢SQL=1 | 慢SQL>=2 |

### 压力与容量预估

首先，由产品同学给出线上交易量预估，并根据TPS线上线下换算公式，得出线下TPS基线。

* **达标目标：TPS基线必须在轻负载区区域内。**

面对数据库或第三方依赖，需要评估可能的瓶颈，数据库最好使用真实的并容量与线上最大值保持一致，因为数据库成为瓶颈的可能性比较大。

假如我们有一个数据库线上有数据100G，分了4个库，每个库128张表，平均每张表有195M数据，约15w条数据。

性能上我们可以只构造一个库的情况，约25G的数据。我们在做性能测试之前必须提早做好这25G的数据。

其他第三方依赖如果已经做过良好的性能评估，比如内部消息系统服务，那么我们可以适当采用性能桩的方式来替代真实后端。

**注：不管是压力还是容量，都离不开具体的业务分析，只能这样才能把性能测试做到最完美。**

## 性能测试准备

### 性能测试环境申请

当做完性能需求分析之后，就要申请性能测试环境。

因为性能测试需要在物理机上运行，而物理机又是稀缺资源，所以我们每次做性能测试都需要提前申请。

目前我们的环境管理维护平台还没建立起来，所以申请就通过邮件形式进行吧。

* **发送申请邮件给[tanghongshan@chinabank.com.cn](mailto:tanghongshan@chinabank.com.cn)，[并抄送给wy-techqa@jd.com](mailto:并抄送给wy-techqa@jd.com)，注明使用机器、服务类型、使用时长、接口人等。**

### 环境清理

在部署系统之前必须要做的一件事就是环境清理，最简单的就是统统删除然后重新搭建一个干净无污染的系统。

如果是在旧系统上做更新，那至少也得把Log日志清理一下、其他可能的干扰进程该杀就杀掉、定时跑的任务、临时文件、初始化文件等等该清理的都清理。

### 环境搭建及数据准备

环境搭建理想的情况是使用我们的环境搭建平台，或者一键式环境搭建脚本。当然，如果都没有的话，我们就得按照我们的上线步骤流程来一步步搭建了。

特别注意当对数据库、消息系统、第三方服务等有依赖的时候，需要及时考虑是采用哪种方案。一是用搭建全套子系统的方式，二是用AK47中的MockServer的解决方案，即部署压力桩。

数据准备更是要提前做，比如为了模拟线上大量数据情况，我们需要灌入3kw订单数据到Oracle。这可能需要花几天时间。

### 压力工具选择

当我们做了性能需求分析、制定了测试方案，这时候需要选取一款合适的性能测试工具，并通过这个工具快速高效的完成测试任务。

通常我们用的压力工具，如：ab、JMeter、LoadRunner等工具，这些在网上都有各种的使用方法，这里就不再一一介绍了。

我们需要了解不同的压力工具的特点。

比如apache的**ab**，它是采用了Linux 2.6内核之后引入的epoll模型，能够制造非常高的压力，尤其是在高并发的环境下最能体现出它的优势。如果我们要压某个耗时稍长的请求，比如某个css静态文件，ab是非常合适的。Ab的缺点是不够灵活。

**JMeter**采用的是多线程模型，扩展性很强，不过制造压力没有那么高。它很适合用来压一些Tomcat服务，或者一些后端接口。JMeter的缺点是压力值不能精确控制，难以适应高并发的情况。

**LoadRunne**r更像是一个模拟器，它比较适用于前端构造较复杂场景的情况，比如模拟100个用户登录的场景，LoadRunner对非技术人员提供了很好的支持。LoadRunner不适用后端接口。

下图为JMeter和LoadRunner对比表：

|  |  |  |
| --- | --- | --- |
| 描述 | JMeter | LoadRunner |
| 架构原理 | 通过中间代理，监控和收集并发客户端的指令，把他们生成脚本，再发送的应用服务器，再监控应用服务器反馈的过程 | 同JMeter |
| 安装 | 简单，解压即可 | LoadRunner安装包有1G多，安装通常要一个多小时，要是装过较旧的盗版还不能再装新版 |
| 支持的协议 | 支持多种协议：HTTP、HTTPS、SOAP、FTP、Database via JDBC、LDAP、JMS、SMTP、POP3、IMAP、MongoDB (NoSQL)、Native commands or shell scripts、TCP等 | 同JMeter |
| 脚本录制 | 提供了一个利用本地ProxyServer（代理服务器）来录制生成测试脚本的功能，也支持badboy录制再生成JMeter脚本 | 自带录制功能强大，可直接录制回放 |
| 并发模型 | 通过增加线程组的数目，或者是设置循环次数来增加并发用户 | 支持多种并发模型，通过在场景中选择要设置什么样的场景，然后选择虚拟用户数 |
| 分布式测试 | 支持，可设置多台代理，通过远程控制实现多台机器并发压力 | 同JMeter |
| 资源监控 | 通过JMeterPlugins插件和ServerAgent实现 | 自带资源监控功能 |
| 报告分析 | 通过与Ant集成，生成HTML报告 | 自身支持生成HTML、Word报告 |
| 虚拟IP | 不支持 | 支持 |
| 网速模拟 | 不支持 | 支持 |
| 扩展性 | 开源，可根据需求修改源码 | 通过扩展函数库实现 |
| 学习成本 | 主要是自学官网上的资料， | 网上资料和相关培训很多，购买正版的话，还有技术支持 |

现在要推荐的是我们**自主研发的AK47框架**中的压力组件，它是基于Java NIO模型，可适用于高并发的情况，制造压力的能力不逊于ab，同时可以用来做压力驱动和压力桩。

使用AK47可以极大的减少准备压力工具的时间，因为压力工具其实就是我们写的自动化case稍加改动而来。

AK47更详细材料请见wiki。

### 资源监控工具部署

Linux系统资源、JVM等监控工具非常多，不过非常遗憾的是很难有一种工具都大面积适用。所以我们计划写一个专用于Linux下Java程序的资源监控脚本。

目前入选的有nmon（Linux系统资源）、jstat、lsof、jmap、jstrace等工具，我们的脚本将这些监控工具综合起来一起。

对数据库也需要监控，错误的SQL、慢查询SQL等都是很重要的线索，具体监控形式还需要与DBA协商。

## 性能测试执行

### 无人值守执行性能测试

无人值守是最理想化的目标，目前我们也朝着这个方向努力。

无人值守不是说没有人力介入，而是把人为的分析和创造性的设计融入到性能测试设计里，至于执行过程只是机器服从指令的运行而已。

通常我们的线下测试环境在白天比较繁忙，各种服务交叉影响，出现性能问题及定位难度较大。

所以**建议性能测试最好在晚上进行**，相对较安静的条件有利于测试结果的稳定性。

### 动态调优

性能测试的吸引力之一就在于它的不可预知性。可能你以前也遇到过，一个服务接口，压力怎么也上不去。费了半天劲，最后才发现是压力工具的问题。

当我们在做压力测试的时候遇到跟预期不符的情况很正常，这个时候需要冷静的分析。问题的种类五花八门，这里无法一一罗列，但万变不离其宗，只要能打破沙锅问到底就没有解决不了的问题。

动态调优的过程其实也是对系统逐渐深入的过程，是后面性能优化的铺垫。

### 边执行边思考

性能测试是一个涉及知识面极广的工作，要成为一个性能测试专家需要不断积累，不停的思考。有些与性能相关的内容也许不会在性能测试报告中体现，但如果我们那么做了，我们会对系统的性能更有信心。

* **用户视角：**
* 还要让我等多久？——响应时间
* 为什么总是失败？——稳定性
* **管理员视角：**
* 服务器资源使用合理吗？——资源利用率
* 数据库使用合理吗？——资源利用率
* 系统能否实现扩展？——可扩展性
* 最多支撑多少用户访问？——系统容量
* 最大业务处理量？——系统容量
* 系统有哪些潜在的瓶颈？——可扩展性
* 更换哪些设备，添加哪些机器可以提高系统性能？——可扩展性
* 7 X 24 小时连续不间断业务访问？——稳定性
* **开发视角：**
* 架构设计是否合理？——架构设计
* 数据库设计是否合理？——数据库设计
* 代码是否存在性能问题？——代码
* 是否有不合理的内存使用？——代码
* 是否有不合理的线程同步操作？——代码
* 是否有不合理的资源竞争？——代码
* 代码算法是否还能有进一步提升？——代码

相信站在不同的角度思考，会有更深的理解。

## 性能分析与调优

关于性能分析与调优这是一个比较大的话题，这里我们单独作为一个章节进行介绍。请阅读第二章性能分析与调优。

## 性能测试报告与总结

性能测试报告是性能测试的里程碑,通过报告能展示出性能测试的最终成果,展示系统性能是否符合需求,是否有性能隐患。性能测试报告中需要阐明性能测试目标、性能测试环境、性能测试数据构造规则、性能测试策略、性能测试结果、性能测试调优说明、性能测试过程中遇到的问题和解决办法等。

具体性能测试报告的编写请下载《性能测试报告模板》。

性能测试工程师完成该次性能测试后，需要将测试结果进行备案，并做为下次性能测试的基线标准，具体包括性能测试结果数据、性能测试瓶颈和调优方案等。同时需要将测试过程中遇到的问题，包括代码瓶颈、配置项问题、数据问题和沟通问题，以及解决办法或解决方案，进行知识沉淀。

## 性能准入标准与建议

性能标准阈值是对指定模块程序性能的基本要求。若达不到该要求，可认为该模块程序的性能达不到上线要求。对不同类型的产品/模块/程序，性能标准阈值的表现形式和数值均有差异。根据产品/模块/程序的差异，分为如下几类：

1.Web交互型：通过浏览器与后台交互并实现各项功能

2.后台接口型：通过接口方式提供服务并实现各项功能

3.离线计算型：通过脚本定时、存储过程的方式实现后台离线计算

针对这几类应用我们给出一些性能准入测试的建议，见下表

|  |  |  |  |
| --- | --- | --- | --- |
| 模块类型 | 指标 | 阈值 | 备注 |
| Web交互型 | 浏览器端操作响应时间 | 90%小于3秒  99%小于10秒 | 包括浏览器+网络+后台接口整体耗时；  网络环境按线上平均带宽预估 |
| 浏览器端渲染性能 | 页面渲染时间<1秒  JS响应时间<1秒 |  |
| 后台资源占用 | CPU Idle 最小值>50%  CPU Idle 均值> 75%  内存占用最大峰值< 80%  内存占用均值<50%  带宽占用<30% |  |
| 后台接口型 | 接口访问响应时间 | 90%概率小于3秒  99%概率小于10秒 | 包括网络+后台接口整体耗时。网络环境按线上平均带宽预估。 |
| 后台资源占用 | CPU Idle 最小值>50%  CPU Idle 均值> 75%  内存占用最大峰值< 80%  内存占用均值<50%  带宽占用<30% | 在正常压力范围的性能场景 |
| 离线计算型 | 计算耗时 | 耗时<运行间隔 |  |
| 资源占用 | CPU Idle 最小值>25%  CPU Idle 均值> 50%  内存占用最大峰值< 80%  内存占用均值<50% |  |

# [性能分析与调优](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1968332)

本章重点介绍性能分析，包括性能数据解读、性能问题发生的原因，如何分析等。对于性能调优，这里只是给一些简单的建议与参考。

## 性能分析

性能结果分析是性能测试中的一个重要部分，同时也是一个难点。由于不同的软件系统，不同的性能指标，结果分析方法都是不一样的。需要具体问题具体分析。下面将阐述一些性能分析的方法与建议。

### 性能分析的目的

1）找出系统瓶颈（硬件、软件）

2）提出性能优化方案

3）达到合理的硬件和软件配置

4）使系统资源使用达到最大平衡

### 常见性能瓶颈征兆

在性能测试执行过程中，我们需要观察和了解系统的运行状态，如果出现以下征兆，则表示系统可能存在瓶颈。

1)持续缓慢：应用程序一直特别慢，改变负载，对整体响应时间影响很少；

2) 随着时间推进越来越慢：负载不变，随着时间推进越来越慢，可能到达某个阈值，系统被锁定或出现大量错误而崩溃；

3)随着负载增加越来越慢：每增加若干用户，系统明显变慢，用户离开系统，系统恢复原状；

4)零星挂起或异常错误：可能是负载或某些原因，用户看到页面无法完成并挂起，无法消除；

5)可预见的锁定：一旦出现挂起或错误，就加速出现，直到系统完全锁定。通常要重启系统才解决。

6)突然混乱：系统一直运行正常，可能是一个小时或三天之后，系统突然出项大量错误或锁定。

### 性能数据解读建议

性能分析过程也是一个解读数据的过程，读懂了数据你就能知道问题出在何处。随着经验的累积将会很容易判断问题的根源，甚至在开发阶段就能对可能出现问题的点打预防针。

|  |  |  |  |
| --- | --- | --- | --- |
| 性能指标类型 | 标准 | 性能瓶颈征兆 | 分析工具 |
| TPS及其波动范围 | 1.Tps符合性能目标  2.Tps轨迹波动平稳 | 1.TPS有明显的大幅波动，不稳定。例如TPS轨迹缓慢下降，缓慢上升后骤降，呈瀑布型，呈矩形，分时间段有规律的波动，无规律的波动等。这些TPS的波动轨迹反映出被测试的性能点存在性能瓶颈，需要性能测试工程师与开发工程师查找性能瓶颈的原因。  2. TPS轨迹比较平稳，但是也存在波动现象。该类波动不明显，很难直接确定是否存在性能瓶颈。我们需要根据其他指标来进行判断。 | Jmeter/loadrunner |
| 响应时间 | 90%平均事务响应时间<性能目标 | 1.关注高峰负载时，用户操作响应时间；   2.关注数据库增量，对用户操作响应时间的影响。 | Jmeter/loadrunner |
| Web\DB服务器内存 |  | 1.很高的换页率  2.进程进入不活动状态;  3.交换区所有磁盘的活动次数过高;  4.过高的全局系统CPU利用率;  5.内存不够出错(out of memory errors) | Nmon/top/vmstat |
| WEB\DB服务器CPU | 合理使用的范围在60%至70% | 1.很慢的响应时间  2.CPU空闲时间为零  3.过高的用户占用CPU时间  4.过高的系统占用CPU时间  5.长时间的有很长的运行进程队列 | Nmon/vmstat/top |
| WEB\DB服务器磁盘I/O | Iowait<30% | 1.过高的磁盘利用率；  2.太长的磁盘等待队列；  3.等待磁盘I/O的时间所占的百分率太高；  4.太高的物理I/O速率；  5.过低的缓存命中率；  6.太长的运行进程队列，但CPU却空闲； | Nmon/sar/iostat |
| Oracle数据库 |  | 1.缓存命中率小于0.90  2.top 10sql耗时高  Oracle数据库的分析和优化，是一门专门的技术，进一步的分析可查相关资料，也可以查看数据库性能分析工具AWR章节。 | AWR |

### 如何定位性能问题

性能问题的定位排查过程比较复杂，可以采用“拆分问题，隔离分析”的方法进行分析，即逐步定位、从外到内、从表及里、逐层分解、隔离排除。以下分析顺序可供参考。

 日志分析--->服务器硬件瓶颈---〉网络瓶颈（对局域网，可以不考虑）---〉服务器操作系统瓶颈（参数配置）---〉中间件瓶颈（参数配置，web服务器等）---〉数据库及应用瓶颈（SQL语句、数据库设计、业务逻辑、算法等）。

以上过程并不是每个分析中都需要的，要根据测试目的和要求来确定分析的深度。整个过程中，要配套使用一些健康工具和日志进行。如： JDK自带的Jconsole，或者JProfiler，来监控服务器性能，oracle的监控工具awr等，具体工具可以参考工具篇。

另外，做性能测试的时候，我们一定要确保瓶颈不要发生在自己的测试脚本和测试工具上。

基于上述思想的指导，在具体执行层面，可以参考如下分析过程：

首先，当我们系统有问题的时候，我们不要急于去调查我们代码，这个毫无意义。我们首要需要看的是操作系统的报告。看看操作系统的CPU利用率，看看内存使用率，看看操作系统的IO，还有网络的IO，网络链接数，等等。通过观察这些数据，我们就可以知道我们的软件的性能基本上出在哪里。比如：

　　1）先看CPU利用率，如果CPU利用率不高，但是系统的Throughput和Latency上不去了，这说明我们的程序并没有忙于计算，而是忙于别的一些事，比如IO。（另外，CPU的利用率还要看内核态的和用户态的，内核态的一上去了，整个系统的性能就下来了。而对于多核CPU来说，CPU 0是相当关键的，如果CPU 0的负载高，那么会影响其它核的性能，因为CPU各核间是需要有调度的，这靠CPU0完成）

　　2）然后，我们可以看一下IO大不大，IO和CPU一般是反着来的，CPU利用率高则IO不大，IO大则CPU就小。关于IO，我们要看三个事，一个是磁盘文件IO，一个是驱动程序的IO（如：网卡），一个是内存换页率。这三个事都会影响系统性能。

　　3）然后，查看一下网络带宽使用情况，在Linux下，你可以使用iftop，iptraf，ntop，tcpdump这些命令来查看。或是用Wireshark来查看。

　　4）如果CPU不高，IO不高，内存使用不高，网络带宽使用不高。但是系统的性能上不去。这说明你的程序有问题，比如，你的程序被阻塞了。可能是因为等那个锁，可能是因为等某个资源，或者是在切换上下文。

通过了解操作系统的性能，我们才知道性能的问题，比如：带宽不够，内存不够，TCP缓冲区不够，等等，很多时候，不需要调整程序的，只需要调整一下硬件或操作系统的配置就可以了。具体配置项的调优，可以参考配置项调优参考章节。

接下来，我们需要使用性能检测工具，也就是使用某个Profiler来差看一下我们程序的运行性能。如：Java的JProfiler/TPTP/CodePro Profiler，GNU的gprof，IBM的PurifyPlus，Intel的VTune，AMD的CodeAnalyst，还有Linux下的OProfile/perf，后面两个可以让你对你的代码优化到CPU的微指令级别，如果你关心CPU的L1/L2的缓存调优，那么你需要考虑一下使用VTune。使用这些Profiler工具，可以让你程序中各个模块函数甚至指令的很多东西，如：运行的时间，调用的次数，CPU的利用率，等等。这些东西对我们来说非常有用。

我们重点观察运行时间最多，调用次数最多的那些函数和指令。这里注意一下，对于调用次数多但是时间很短的函数，你可能只需要轻微优化一下，你的性能就上去了（比如：某函数一秒种被调用100万次，你想想如果你让这个函数提高0.01毫秒的时间，这会给你带来多大的性能）

使用Profiler有个问题我们需要注意一下，因为Profiler会让你的程序运行的性能变低，像PurifyPlus这样的工具会在你的代码中插入很多代码，会导致你的程序运行效率变低，从而没发测试出在高吞吐量下的系统的性能，对此，一般有两个方法来定位系统瓶颈：

1）在你的代码中自己做统计，使用微秒级的计时器和函数调用计算器，每隔10秒把统计log到文件中。

2）分段注释你的代码块，让一些函数空转，做Hard Code的Mock，然后再测试一下系统的Throughput和Latency是否有质的变化，如果有，那么被注释的函数就是性能瓶颈，再在这个函数体内注释代码，直到找到最耗性能的语句。

最后再说一点，对于性能测试，不同的Throughput会出现不同的测试结果，不同的测试数据也会有不同的测试结果。所以，用于性能测试的数据非常重要，性能测试中，我们需要观测试不同Throughput的结果。

### 常见性能问题参考

下面是整理收集的一些常见问题列表，不全，也许并不对，大家可以补充指正。

|  |  |
| --- | --- |
| 类别 | 常见性能问题 |
| 操作系统类 | 1. Sys的CPU使用率过高  2. User的CPU使用率过高，持续大于80%以上  3.可用物理内存不足导致内存溢出  4. 磁盘空间不足导致交易处理失败，性能下降  5. TCP/IP连接数限制导致用户请求失败  6.磁盘IO使用比较繁忙，持续大于70% |
| 中间件类 | 常用主流中间件：Tomcat、apache、nginx、Weblogic、Jboss等  1.线程不回收导致溢出，引发宕机  2.数据库连接池不释放导致溢出  3.JVM内存参数设置不合理，新生代过大或偏小  4.永久代设置过小，导致栈溢出  5.其它问题 |
| 应用程序类 | 1.程序响应时间超长  2. JAVA程序内存溢出，内存中存放大量数据对象  3. JAVA程序循环嵌套过多，过于精细的查询条件，子查询间等待超时  4.程序中存在死循环引起线程死锁，导致CPU使用率达到100%  5.某些返回结果未定义处理方式，导致线程等待，不释放，CPU使用率高 |
| 数据库类 | 1.SGA分配不合理，需要具体情况具体分析  2.使用全表扫描  3.对于查询业务比较多的表，未建立索引，或建立的索引不合理，在索引列上使用IS NULL和IS NOT NULL  4.存在数据库死锁导致数据库连接超时或不释放。  5.存在过于复杂的计算，导致CPU、内存和IO使用率较高。  6. 数据库读写过于频繁，导致IO使用率比较高 |
| 其他问题 | 1.网络问题，被测试环境网络环境小于100M  2. 客户端问题等等 |

## 性能调优

性能调优是一个非常大的议题，更多的是开发人员进行，对于测试人员，可以了解一些通用的调优方法，并根据性能分析过程中发现的问题，给出一些建议。

### 性能调优目标与策略

性能优化的目标不外乎两个：1.时间性能：减小系统执行的时间；2.空间性能：减小系统占用的空间。

一般来说，性能优化也就是下面的几个策略：

用空间换时间。各种cache如CPU L1/L2/RAM到硬盘，都是用空间来换时间的策略。这样策略基本上是把计算的过程一步一步的保存或缓存下来，这样就不用每次用的时候都要再计算一遍，比如数据缓冲，CDN，等。这样的策略还表现为冗余数据，比如数据镜象，负载均衡什么的。

用时间换空间。有时候，少量的空间可能性能会更好，比如网络传输，如果有一些压缩数据的算法，这样的算法其实很耗时，但是因为瓶颈在网络传输，所以用时间来换空间反而能省时间。

简化代码。最高效的程序就是不执行任何代码的程序，所以，代码越少性能就越高。如：减少循环的层数，减少递归，在循环中少声明变量，少做分配和释放内存的操作，尽量把循环体内的表达式抽到循环外，条件表达的中的多个条件判断的次序，尽量在程序启动时把一些东西准备好，注意函数调用的开销（栈上开销），注意面向对象语言中临时对象的开销，小心使用异常，等等，这连东西需要我们非常了解编程语言和常用的库。

并行处理。如果CPU只有一个核，你要玩多进程，多线程，对于计算密集型的软件会反而更慢（因为操作系统调度和切换开销很大），CPU的核多了才能真正体现出多进程多线程的优势。并行处理需要我们的程序有Scalability，不能水平或垂直扩展的程序无法进行并行处理。从架构上来说，是否可以做到不改代码只是加加机器就可以完成性能提升？

总之，根据2：8原则来说，20%的代码耗了你80%的性能，找到那20%的代码，你就可以优化那80%的性能。

### 调优的原则

* 在应用系统的设计、开发过程中，应始终把性能放在考虑的范围内。
* 确定清晰明确的性能目标是关键。
* 必须保证调优后的程序运行正确。
* 性能更大程度是取决于良好的设计，调优技巧只是一个辅助手段。
* 调优过程是叠代渐进的过程，每次调优的结果要反馈到后续的代码开发中去。
* 性能调优不能以牺牲代码的可读性和维护性为代价。

### 调优的基本步骤

1. 确定清晰的性能目标，并按优先级排列
2. 利用科学的测试工具对应用程序进行测试，并记录测试结果。
3. 把分布式系统拆分成组件：Web层、业务层、集成层、以及网络传输时间，分别进行调优。
4. 有系统的科学调优
   * 遵循一定的程序：测试性能→找出瓶颈→假设造成瓶颈的因素→测试假设是否成立→修改应用→再次测试性能
   * 确定影响性能的因素：CPU、内存还是IO。
   * 找出主要的瓶颈，首先解决最容易的，再重复测试。
   * 一次修改一个瓶颈，不要对不需要的地方进行调优
   * 提高CPU性能：更快的代码，更好的算法，减少短期生存的对象。
   * 提高内存性能：减少或减小长期生存的对象。
   * 提高IO性能：重新设计应用，减少IO的交互。

5.优化完成之后，进行QA测试。

6.在代码中记录优化的地方，并对旧代码进行注释。

### 性能调优的层次

为了提升系统性能，开发人员可以从系统的各个角度和层次对系统进行优化。除了最常见的代码优化外，在软件架构上、JVM虚拟机层、数据库以及操作系统层面都可以通过各种手段进行调优，从而在整体上提升系统的性能。

#### 设计调优

设计调优处于所有调优手段的上层，它往往需要在软件开发之前进行。在软件开发之初，架构师就应该评估系统可能存在的各种潜在问题，并给出合理的设计方案。由于软件设计和架构对软件整体质量有决定性的影响，所以，设计调优对系统性能的影响也是最大的。如果说，代码优化、JVM优化都是对系统微观层面上"量"的优化，那么设计优化就是对系统在宏观层面上"质"的优化。

设计优化的一大显著特点是，它可以规避某一个组件的性能问题，而非改良该组件的实现。比如，系统中组件A需要等待某事件E才能触发一个行为。如果组件A通过循环监控不断监测事件E是否发生，其监测行为必然会占用部分系统资源，因此，开发人员必须在监测频率和资源消耗间取得平衡。如果监测频率太低，虽然减少了资源消耗，但是系统实时反应性就会降低。如果进行代码层的调优，就需要优化监测方法的实现以及求得一个最为恰当的监测频率。

而若将此问题预留在设计层解决，便可以使用事件通知的方式将系统行为进行倒置。如使用第2章中提到的观察者模式，在事件E发生的时刻，由事件E通知组件A，从而触发组件A的行为。这种设计方法弃用了存在性能隐患的循环监控，从根本上解决了这一问题。

从某种程度上说，设计优化直接决定了系统的整体品质。如果在设计层考虑不周，留下太多问题隐患，那么这些"质"上的问题，也许无法再通过代码层的优化进行弥补。因此，开发人员必须在软件设计之初，认真仔细考虑软件系统的性能问题。

进行设计优化时，设计人员必须熟悉常用的软件设计方法、设计模式、基本性能组件和常用优化思想，并将其有机地集成在软件系统中。

注意：一个良好的系统设计可以规避很多潜在的性能问题。因此，尽可能多花些时间在系统设计上，是创建高性能程序的关键。

#### 算法调优

算法非常重要，好的算法会有更好的性能。举几个例子，大家可以感觉一下。

一个是过滤算法。系统需要对收到的请求做过滤，我们把可以被filter in/out的东西配置在了一个文件中，原有的过滤算法是遍历过滤配置，后来，我们找到了一种方法可以对这个过滤配置进行排序，这样就可以用二分折半的方法来过滤，系统性能增加了50%。

　　一个是哈希算法。计算哈希算法的函数并不高效，一方面是计算太费时，另一方面是碰撞太高，碰撞高了就跟单向链表一个性能（可参看Hash Collision DoS 问题）。我们知道，算法都是和需要处理的数据很有关系的，就算是被大家所嘲笑的“冒泡排序”在某些情况下（大多数数据是排好序的）其效率会高于所有的排序算法。哈希算法也一样，广为人知的哈希算法都是用英文字典做测试，但是我们的业务在数据有其特殊性，所以，对于还需要根据自己的数据来挑选适合的哈希算法。

　　分而治之和预处理。如某程序为了生成月报表，每次都需要计算很长的时间，有时候需要花将近一整天的时间。于是我们把我们找到了一种方法可以把这个算法发成增量式的，也就是说我每天都把当天的数据计算好了后和前一天的报表合并，这样可以大大的节省计算时间，每天的数据计算量只需要20分钟，但是如果我要算整个月的，系统则需要10个小时以上（SQL语句在大数据量面前性能成级数性下降）。这种分而治之的思路在大数据面前对性能有很帮助，就像merge排序一样。SQL语句和数据库的性能优化也是这一策略，如：使用嵌套式的Select而不是笛卡尔积的Select，使用视图，等等。

#### 代码调优

代码调优是在软件开发过程中，或者在软件开发完成后，软件维护过程中进行的对程序代码的改进和优化。代码优化涉及诸多编码技巧，需要开发人员熟悉相关语言的API，并在合适的场景中正确使用相关API或类库。同时，对算法、数据结构的灵活使用，也是代码优化的重要内容。

虽然代码优化是从微观上对性能进行调整，但是一个"好"的实现和一个"坏"的实现对系统的影响也是非常大的。比如，同样作为List的实现，LinkedList和ArrayList在随机访问上的性能却可以相差几个数量级；又如，同样是文件读写的实现，使用Stream方式与Java NIO的方式，其性能可能又会相差一个数量级。

因此，虽然与设计优化相比，这里将代码优化称为在微观层面上的优化，但是它却是对系统性能产生最直接影响的优化方法。

#### JVM调优

由于Java软件总是运行在JVM虚拟机之上，对JVM虚拟机进行优化也能在一定程度上提升Java程序的性能。JVM调优通常可以在软件开发后期进行，如在软件开发完成，或者在软件开发的某一里程碑阶段。

作为Java软件的运行平台，JVM的各项参数将会直接影响Java程序的性能。比如，JVM的堆大小、垃圾回收策略等。

要进行JVM层面的调优，需要开发人员对JVM的运行原理和基本内存结构有一定了解。如，堆内存的结构、GC的种类等。然后，依据应用程序的特点，设置合理的JVM启动参数。

#### 网络调优

　　关于网络调优，尤其是TCP Tuning，这里面有很多很多东西可以说。看看Linux下TCP/IP的那么多参数就知道了。强烈建议大家看看《TCP/IP详解卷1:协议》这本书。在这里只讲一些概念上的东西。

　　A)TCP调优

　　我们知道TCP链接是有很多开销的，一个是会占用文件描述符，另一个是会开缓存，一般来说一个系统可以支持的TCP链接数是有限的，我们需要清楚地认识到TCP链接对系统的开销是很大的。正是因为TCP是耗资源的，所以，很多攻击都是让你系统上出现大量的TCP链接，把你的系统资源耗尽。比如著名的SYNC Flood攻击。所以，我们要注意配置KeepAlive参数，这个参数的意思是定义一个时间，如果链接上没有数据传输，系统会在这个时间发一个包，如果没有收到回应，那么TCP就认为链接断了，然后就会把链接关闭，这样可以回收系统资源开销。（注：HTTP层上也有KeepAlive参数）对于像HTTP这样的短链接，设置一个1-2分钟的keepalive非常重要。这可以在一定程度上防止DoS攻击。有下面几个参数（下面这些参数的值仅供参考）：

　　net.ipv4.tcp\_keepalive\_probes = 5

　　net.ipv4.tcp\_keepalive\_intvl = 20

　　net.ipv4.tcp\_fin\_timeout = 30

　　对于TCP的TIME\_WAIT这个状态，主动关闭的一方进入TIME\_WAIT状态，TIME\_WAIT状态将持续2个MSL(Max Segment Lifetime)，默认为4分钟，TIME\_WAIT状态下的资源不能回收。有大量的TIME\_WAIT链接的情况一般是在HTTP服务器上。对此，有两个参数需要注意，

　　net.ipv4.tcp\_tw\_reuse=1

　　net.ipv4.tcp\_tw\_recycle=1

　　前者表示重用TIME\_WAIT，后者表示回收TIME\_WAIT的资源。

　　TCP还有一个重要的概念叫RWIN（TCP Receive Window Size），这个东西的意思是，一个TCP链接在没有向Sender发出ack时可以接收到的最大的数据包。为什么这个很重要？因为如果Sender没有收到Receiver发过来ack，Sender就会停止发送数据并会等一段时间，如果超时，那么就会重传。这就是为什么TCP链接是可靠链接的原因。重传还不是最严重的，如果有丢包发生的话，TCP的带宽使用率会马上受到影响（会盲目减半），再丢包，再减半，然后如果不丢包了，就逐步恢复。相关参数如下：

　　net.core.wmem\_default = 8388608

　　net.core.rmem\_default = 8388608

　　net.core.rmem\_max = 16777216

　　net.core.wmem\_max = 16777216

　　一般来说，理论上的RWIN应该设置成：吞吐量\*回路时间。Sender端的buffer应该和RWIN有一样的大小，因为Sender端发送完数据后要等Receiver端确认，如果网络延时很大，buffer过小了，确认的次数就会多，于是性能就不高，对网络的利用率也就不高了。也就是说，对于延迟大的网络，我们需要大的buffer，这样可以少一点ack，多一些数据，对于响应快一点的网络，可以少一些buffer。因为，如果有丢包（没有收到ack），buffer过大可能会有问题，因为这会让TCP重传所有的数据，反而影响网络性能。（当然，网络差的情况下，就别玩什么高性能了）所以，高性能的网络重要的是要让网络丢包率非常非常地小（基本上是用在LAN里），如果网络基本是可信的，这样用大一点的buffer会有更好的网络传输性能（来来回回太多太影响性能了）。

　　另外，我们想一想，如果网络质量非常好，基本不丢包，而业务上我们不怕偶尔丢几个包，如果是这样的话，那么，我们为什么不用速度更快的UDP呢？你想过这个问题了吗？

　　B)UDP调优

　　说到UDP的调优，有一些事我想重点说一样，那就是MTU——最大传输单元（其实这对TCP也一样，因为这是链路层上的东西）。所谓最大传输单元，你可以想像成是公路上的公交车，假设一个公交车可以最多坐70人，带宽就像是公路的车道数一样，如果一条路上最多可以容下100辆公交车，那意味着我最多可以运送7000人，但是如果公交车坐不满，比如平均每辆车只有20人，那么我只运送了2000人，于是我公路资源（带宽资源）就被浪费了。所以，我们对于一个UDP的包，我们要尽量地让他大到MTU的最大尺寸再往网络上传，这样可以最大化带宽利用率。对于这个MTU，以太网是1500字节，光纤是4352字节，802.11无线网是7981。但是，当我们用TCP/UDP发包的时候，我们的有效负载Payload要低于这个值，因为IP协议会加上20个字节，UDP会加上8个字节（TCP加的更多），所以，一般来说，你的一个UDP包的最大应该是1500-8-20=1472，这是你的数据的大小。当然，如果你用光纤的话，这个值就可以更大一些。（顺便说一下，对于某些NB的千光以态网网卡来说，在网卡上，网卡硬件如果发现你的包的大小超过了MTU，其会帮你做fragment，到了目标端又会帮你做重组，这就不需要你在程序中处理了）

　　再多说一下，使用Socket编程的时候，你可以使用setsockopt() 设置SO\_SNDBUF/SO\_RCVBUF的大小，TTL和KeepAlive这些关键的设置，当然，还有很多，具体你可以查看一下Socket的手册。

　　最后说一点，UDP还有一个最大的好处是multi-cast多播，这个技术对于你需要在内网里通知多台结点时非常方便和高效。而且，多播这种技术对于机会的水平扩展（需要增加机器来侦听多播信息）也很有利。

　　C）网卡调优

　　对于网卡，我们也是可以调优的，这对于千兆以及网网卡非常必要，在Linux下，我们可以用ifconfig查看网上的统计信息，如果我们看到overrun上有数据，我们就可能需要调整一下txqueuelen的尺寸（一般默认为1000），我们可以调大一些，如：ifconfig eth0 txqueuelen 5000。Linux下还有一个命令叫：ethtool可以用于设置网卡的缓冲区大小。在Windows下，我们可以在网卡适配器中的高级选项卡中调整相关的参数（如：Receive Buffers, Transmit Buffer等，不同的网卡有不同的参数）。把Buffer调大对于需要大数据量的网络传输非常有效。

　　D）其它网络性能

　　关于多路复用技术，也就是用一个线程来管理所有的TCP链接，有三个系统调用要重点注意：一个是select，这个系统调用只支持上限1024个链接，第二个是poll，其可以突破1024的限制，但是select和poll本质上是使用的轮询机制，轮询机制在链接多的时候性能很差，因主是O(n)的算法，所以，epoll出现了，epoll是操作系统内核支持的，仅当在链接活跃时，操作系统才会callback，这是由操作系统通知触发的，但其只有Linux Kernel 2.6以后才支持（准确说是2.5.44中引入的），当然，如果所有的链接都是活跃的，过多的使用epoll\_ctl可能会比轮询的方式还影响性能，不过影响的不大。

　　另外，关于一些和DNS Lookup的系统调用要小心，比如：gethostbyaddr/gethostbyname，这个函数可能会相当的费时，因为其要到网络上去找域名，因为DNS的递归查询，会导致严重超时，而又不能通过设置什么参数来设置time out，对此你可以通过配置hosts文件来加快速度，或是自己在内存中管理对应表，在程序启动时查好，而不要在运行时每次都查。另外，在多线程下面，gethostbyname会一个更严重的问题，就是如果有一个线程的gethostbyname发生阻塞，其它线程都会在gethostbyname处发生阻塞，这个比较变态，要小心。这种到网上找信息的东西很多，比如，如果你的Linux使用了NIS，或是NFS，某些用户或文件相关的系统调用就很慢，所以要小心。

#### 系统调优

　　A）I/O模型

　　前面说到过select/poll/epoll这三个系统调用，我们都知道，Unix/Linux下把所有的设备都当成文件来进行I/O，所以，那三个操作更应该算是I/O相关的系统调用。说到I/O模型，这对于我们的I/O性能相当重要，我们知道，Unix/Linux经典的I/O方式是（关于Linux下的I/O模型，大家可以读一下这篇文章《使用异步I/O大大提高性能》）：

　　第一种，同步阻塞式I/O，这个不说了。

　　第二种，同步无阻塞方式。其通过fctnl设置O\_NONBLOCK来完成。

　　第三种，对于select/poll/epoll这三个是I/O不阻塞，但是在事件上阻塞，算是：I/O异步，事件同步的调用。

　　第四种，AIO方式。这种I/O模型是一种处理与I/O并行的模型。I/O请求会立即返回，说明请求已经成功发起了。在后台完成I/O操作时，向应用程序发起通知，通知有两种方式：一种是产生一个信号，另一种是执行一个基于线程的回调函数来完成这次I/O处理过程。

　　第四种因为没有任何的阻塞，无论是I/O上，还是事件通知上，所以，其可以让你充分地利用CPU，比起第二种同步无阻塞好处就是，第二种要你一遍一遍地去轮询。Nginx之所所以高效，是其使用了epoll和AIO的方式来进行I/O的。

　　当然，真正提高I/O性能方式是把和外设的I/O的次数降到最低，最好没有，所以，对于读来说，内存cache通常可以从质上提升性能，因为内存比外设快太多了。对于写来说，cache住要写的数据，少写几次，但是cache带来的问题就是实时性的问题，也就是latency会变大，我们需要在写的次数上和相应上做权衡。

　　B）多核CPU调优

　　关于CPU的多核技术，我们知道，CPU0是很关键的，如果0号CPU被用得过狠的话，别的CPU性能也会下降，因为CPU0是有调整功能的，所以，我们不能任由操作系统负载均衡，因为我们自己更了解自己的程序，所以，我们可以手动地为其分配CPU核，而不会过多地占用CPU0，或是让我们关键进程和一堆别的进程挤在一起。

　　对于Linux来说，可以使用taskset命令来设置（你可以通过安装schedutils来安装这个命令：apt-get install schedutils）

　　多核CPU还有一个技术叫NUMA技术（Non-Uniform Memory Access）。传统的多核运算是使用SMP(Symmetric Multi-Processor )模式，多个处理器共享一个集中的存储器和I/O总线。于是就会出现一致存储器访问的问题，一致性通常意味着性能问题。NUMA模式下，处理器被划分成多个node，每个node有自己的本地存储器空间。关于NUMA的一些技术细节，你可以查看一下这篇文章《Linux的NUMA技术》，在Linux下，对NUMA调优的命令是：numactl 。如下面的命令：（指定命令“myprogram arg1 arg2”运行在node 0上，其内存分配在node 0 和1上）

　　numactl --cpubind=0 --membind=0,1 myprogram arg1 arg2

　　当然，上面这个命令并不好，因为内存跨越了两个node，这非常不好。最好的方式是只让程序访问和自己运行一样的node，如：

　　$ numactl --membind 1 --cpunodebind 1 --localalloc myapplication

　　C）文件系统调优

　　关于文件系统，因为文件系统也是有cache的，所以，为了让文件系统有最大的性能。首要的事情就是分配足够大的内存，这个非常关键，在Linux下可以使用free命令来查看 free/used/buffers/cached，理想来说，buffers和cached应该有40%左右。然后是一个快速的硬盘控制器，SCSI会好很多。最快的是Intel SSD固态硬盘，速度超快，但是写次数有限。

　　接下来，我们就可以调优文件系统配置了，对于Linux的Ext3/4来说，几乎在所有情况下都有所帮助的一个参数是关闭文件系统访问时间，在/etc/fstab下看看你的文件系统有没有noatime参数（一般来说应该有），还有一个是dealloc，它可以让系统在最后时刻决定写入文件发生时使用哪个块，可优化这个写入程序。还要注间一下三种日志模式：data=journal、data=ordered和data=writeback。默认设置data=ordered提供性能和防护之间的最佳平衡。

　　当然，对于这些来说，ext4的默认设置基本上是最佳优化了。

　　其它还有一些关于NFS、XFS的调优，大家可以上google搜索一些相关优化的文章看看。关于各文件系统，大家可以看一下这篇文章——《Linux日志文件系统及性能分析》。

#### 数据库调优

对绝大部分应用系统而言，数据库是必不可少的一部分。Java程序可以使用JDBC的方式连接数据库。对数据库的调优可以分为3个部分：在应用层对SQL语句进行优化；对数据库进行优化；对数据库软件进行优化。

数据库调优是一个很大的话题，下面的这些东西并不一定正确，因为在不同的业务场景，不同的数据库设计下可能会得到完全相反的结论，所以，在这里做一些一般性的说明，具体问题还要具体分析。

　　A）数据库引擎调优

　　数据库的锁的方式。这个非常非常地重要。并发情况下，锁是非常非常影响性能的。各种隔离级别，行锁，表锁，页锁，读写锁，事务锁，以及各种写优先还是读优先机制。性能最高的是不要锁，所以，分库分表，冗余数据，减少一致性事务处理，可以有效地提高性能。NoSQL就是牺牲了一致性和事务处理，并冗余数据，从而达到了分布式和高性能。

　　数据库的存储机制。不但要搞清楚各种类型字段是怎么存储的，更重要的是数据库的数据存储方式，是怎么分区的，是怎么管理的，比如Oracle的数据文件，表空间，段，等等。了解清楚这个机制可以减轻很多的I/O负载。比如：MySQL下使用show engines;可以看到各种存储引擎的支持。不同的存储引擎有不同的侧重点，针对不同的业务或数据库设计会让你有不同的性能。

　　数据库的分布式策略。最简单的就是复制或镜像，需要了解分布式的一致性算法，或是主主同步，主从同步。通过了解这种技术的机理可以做到数据库级别的水平扩展。

　　B）SQL语句优化

　　关于SQL语句的优化，首先也是要使用工具，比如：MySQL SQL Query Analyzer，Oracle SQL Performance Analyzer，或是微软SQL Query Analyzer，基本上来说，所有的RMDB都会有这样的工具，来让你查看你的应用中的SQL的性能问题。 还可以使用explain来看看SQL语句最终Execution Plan会是什么样的。

　　还有一点很重要，数据库的各种操作需要大量的内存，所以服务器的内存要够，优其应对那些多表查询的SQL语句，那是相当的耗内存。

　　下面我根据我有限的数据库SQL的知识说几个会有性能问题的SQL：

　　全表检索。比如：select \* from user where lastname = “xxxx”，这样的SQL语句基本上是全表查找，线性复杂度O(n)，记录数越多，性能也越差（如：100条记录的查找要50ms，一百万条记录需要5分钟）。对于这种情况，我们可以有两种方法提高性能：一种方法是分表，把记录数降下来，另一种方法是建索引（为lastname建索引）。索引就像是key-value的数据结构一样，key就是where后面的字段，value就是物理行号，对索引的搜索复杂度是基本上是O(log(n)) ——用B-Tree实现索引（如：100条记录的查找要50ms，一百万条记录需要100ms）。

　　索引。对于索引字段，最好不要在字段上做计算、类型转换、函数、空值判断、字段连接操作，这些操作都会破坏索引原本的性能。当然，索引一般都出现在Where或是Order by字句中，所以对Where和Order by子句中的子段最好不要进行计算操作，或是加上什么NOT之类的，或是使用什么函数。

　　多表查询。关系型数据库最多的操作就是多表查询，多表查询主要有三个关键字，EXISTS，IN和JOIN（关于各种join，可以参看图解SQL的Join一文）。基本来说，现代的数据引擎对SQL语句优化得都挺好的，JOIN和IN/EXISTS在结果上有些不同，但性能基本上都差不多。有人说，EXISTS的性能要好于IN，IN的性能要好于JOIN，我各人觉得，这个还要看你的数据、schema和SQL语句的复杂度，对于一般的简单的情况来说，都差不多，所以千万不要使用过多的嵌套，千万不要让你的SQL太复杂，宁可使用几个简单的SQL也不要使用一个巨大无比的嵌套N级的SQL。还有人说，如果两个表的数据量差不多，Exists的性能可能会高于In，In可能会高于Join，如果这两个表一大一小，那么子查询中，Exists用大表，In则用小表。这个，我没有验证过，放在这里让大家讨论吧。另，有一篇关于SQL Server的文章大家可以看看《IN vs JOIN vs EXISTS》

　　JOIN操作。有人说，Join表的顺序会影响性能，只要Join的结果集是一样，性能和join的次序无关。因为后台的数据库引擎会帮我们优化的。Join有三种实现算法，嵌套循环，排序归并，和Hash式的Join。（MySQL只支持第一种）

　　（1）嵌套循环，就好像是我们常见的多重嵌套循环。注意，前面的索引说过，数据库的索引查找算法用的是B-Tree，这是O(log(n))的算法，所以，整个算法复法度应该是O(log(n)) \* O(log(m))这样的。

　　（2）Hash式的Join，主要解决嵌套循环的O(log(n))的复杂，使用一个临时的hash表来标记。

　　（3）排序归并，意思是两个表按照查询字段排好序，然后再合并。当然，索引字段一般是排好序的。

　　还是那句话，具体要看什么样的数据，什么样的SQL语句，你才知道用哪种方法是最好的。

　　部分结果集。我们知道MySQL里的Limit关键字，Oracle里的rownum，SQL Server里的Top都是在限制前几条的返回结果。这给了我们数据库引擎很多可以调优的空间。一般来说，返回top n的记录数据需要我们使用order by，注意在这里我们需要为order by的字段建立索引。有了被建索引的order by后，会让我们的select语句的性能不会被记录数的所影响。使用这个技术，一般来说我们前台会以分页方式来显现数据，Mysql用的是OFFSET，SQL Server用的是FETCH NEXT，这种Fetch的方式其实并不好是线性复杂度，所以，如果我们能够知道order by字段的第二页的起始值，我们就可以在where语句里直接使用>=的表达式来select，这种技术叫seek，而不是fetch，seek的性能比fetch要高很多。

　　字符串。正如我前面所说的，字符串操作对性能上有非常大的恶梦，所以，能用数据的情况就用数字，比如：时间，工号，等。

　　全文检索。千万不要用Like之类的东西来做全文检索，如果要玩全文检索，可以尝试使用Sphinx。

　　其它。

　　（1）不要select \*，而是明确指出各个字段，如果有多个表，一定要在字段名前加上表名，不要让引擎去算。

　　（2）不要用Having，因为其要遍历所有的记录。性能差得不能再差。

　　（3）尽可能地使用UNION ALL 取代UNION。

　　（4）索引过多，insert和delete就会越慢。而update如果update多数索引，也会慢，但是如果只update一个，则只会影响一个索引表。

### 配置项性能调优参考

下面的推荐调优配置都是基于下面的机器配置：

|  |  |
| --- | --- |
| OS | CentOS release 6.3 (Final) |
| IP | 192.168.197.138 |
| CPU | 4 核 3000.035 MHz |
| MEMORY | 16G |
| Additional info. | 虚拟机 |

1. Linux系统调优
2. Linux limit 配置修改

|  |
| --- |
| 修改limit 配置文件，提高linux 本身最大限制，都是在文件最后添加，如果有则是修改  vi /etc/security/limits.conf  \* soft nproc 102400000  \* hard nproc 102400000  \* soft nofile 1048576  \* hard nofile 1048576  vi /etc/security/limits.d/90-nproc.conf  \* soft nproc 102400000  \* hard nproc 102400000  \* soft nofile 1048576  \* hard nofile 1048576  重启生效  service sshd restart  为每个普通用户修改limit 限制(经过尝试，两台虚拟机最大的)：  vi /etc/profile  ulimit -u 102400000  ulimit -n 1048576  ulimit -d unlimited  ulimit -m unlimited  ulimit -s unlimited  ulimit -t unlimited  ulimit -v unlimited  ulimit -c unlimited |

1. TCP linux 内核参数设置

|  |
| --- |
| 修改配置文件 /etc/sysctl.conf  vi /etc/sysctl.conf  # Controls the use of TCP syncookies  net.ipv4.tcp\_syncookies = 1  net.ipv4.tcp\_tw\_reuse = 1  net.ipv4.tcp\_tw\_recycle = 1  net.ipv4.tcp\_fin\_timeout = 5  net.ipv4.tcp\_keepalive\_time = 1200  net.ipv4.ip\_local\_port\_range = 10000 65000  net.ipv4.tcp\_max\_syn\_backlog = 262144  net.core.netdev\_max\_backlog = 262144  net.ipv4.tcp\_max\_tw\_buckets = 5000  生效(执行会报错，需要执行中间两个命令)  /sbin/sysctl -p  modprobe bridge  lsmod|grep bridge  /sbin/sysctl –p |

1. **JVM 调优**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **推荐参数：**  JVM\_ARGS=  "-Xms4000M –Xmx4000M –Xmn1000M -XX:PermSize=1000M -XX:MaxPermSize=1000M \  -XX:+UseConcMarkSweepGC-XX:MaxTenuringThreshold=7 -XX:GCTimeRatio=19 \  -Xnoclassgc -Xloggc:log/gc.log -XX:+PrintGCDetails -XX:+PrintGCTimeStamps \  -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=0 \  JVM 堆内存和 GC回收设置，堆内存大小推荐机器物理内存的 1/4, GC 回收用的是CMS内存并发回收。  **堆参数解释：**   |  |  |  | | --- | --- | --- | | 参数名称  含义 | 默认值 |  | | -Xms  初始堆大小 | 物理内存的1/64(<1GB) | 默认(MinHeapFreeRatio参数可以调整)空余堆内存小于40%时，JVM就会增大堆直到-Xmx的最大限制. | | -Xmx  最大堆大小 | 物理内存的1/4(<1GB) | 默认(MaxHeapFreeRatio参数可以调整)空余堆内存大于70%时，JVM会减少堆直到 -Xms的最小限制 | | -Xmn  年轻代大小(1.4or lator) |  | 注意：此处的大小是（eden+ 2 survivor space).与jmap -heap中显示的New gen是不同的。 | | 整个堆大小=年轻代大小 + 年老代大小 + 持久代大小. | | 增大年轻代后,将会减小年老代大小.此值对系统性能影响较大,Sun官方推荐配置为整个堆的3/8 | | -XX:NewSize  设置年轻代大小(for 1.3/1.4) |  |  | | -XX:MaxNewSize  年轻代最大值(for 1.3/1.4) |  |  | | -XX:PermSize  设置持久代(perm gen)初始值 | 物理内存的1/64 |  | | -XX:MaxPermSize  设置持久代最大值 | 物理内存的1/4 |  | |

1. **Tomcat调优**
   1. 增加JVM堆内存大小

|  |
| --- |
| JAVA\_OPTS="-Djava.awt.headless=true -Dfile.encoding=UTF-8  -server –Xms4000m –Xmx4000m  -XX:NewSize=1000m -XX:MaxNewSize=1000m -XX:PermSize=1000m  -XX:MaxPermSize=1000m -XX:+DisableExplicitGC"  -Xms – 指定初始化时化的栈内存, 推荐机器物理内存的 1/4  -Xmx – 指定最大栈内存, 推荐机器物理内存的 1/4  在重启你的Tomcat服务器之后，这些配置的更改才会有效。 |

* 1. 修复JRE内存泄漏

|  |
| --- |
| <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" /> |

* 1. 线程池设置

|  |
| --- |
| <Connector port="8080" address="localhost"  **maxThreads**="250" maxHttpHeaderSize="8192"  emptySessionPath="true" protocol="HTTP/1.1"  enableLookups="false" redirectPort="8181"acceptCount="100"  connectionTimeout="20000" disableUploadTimeout="true"/> |

* 1. 压缩

|  |
| --- |
| <Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8181" **compression**="500"  compressableMimeType="text/html,text/xml,text/plain,application/octet-stream" /> |

1. **Nginx 调优**

|  |
| --- |
| worker\_processes 8;  nginx进程数，建议按照cpu数目来指定，一般为它的倍数, 2倍左右比较合适。  worker\_cpu\_affinity 00000001 00000010 00000100 00001000 00000001 00000010 00000100 00001000;  为每个进程分配cpu，上例中将8个进程分配到4个cpu，当然可以写多个，或者将一个进程分配到多个cpu。  worker\_rlimit\_nofile 102400;  这个指令是指当一个nginx进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx进程数相除，但是nginx分配请求并不是那么均匀，所以最好与ulimit -n的值保持一致。  use epoll;  使用epoll的I/O模型， epoll 是 linux 内核对AIO的实现。  worker\_connections 102400;  每个进程允许的最多连接数，理论上每台nginx服务器的最大连接数为worker\_processes\*worker\_connections。  keepalive\_timeout 60;  keepalive超时时间。  client\_header\_buffer\_size 4k;  客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求的头部大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。分页大小可以用命令getconf PAGESIZE取得。  open\_file\_cache max=102400 inactive=20s;  这个将为打开文件指定缓存，默认是没有启用的，max指定缓存数量，建议和打开文件数一致，inactive是指经过多长时间文件没被请求后删除缓存。  open\_file\_cache\_valid 30s;  这个是指多长时间检查一次缓存的有效信息。  open\_file\_cache\_min\_uses 1;  open\_file\_cache指令中的inactive参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive时间内一次没被使用，它将被移除。 |

1. **MySQLInnoDB调优**

|  |
| --- |
| #如果将back\_log设定到比你操作系统限制更高的值,将会没有效果,  #对应 Linux tcp 内核参数中的net.core.netdev\_max\_backlog  back\_log = 30000  # MySQL 服务所允许的同时会话数的上限  max\_connections = 3000  # 每个客户端连接最大的错误允许数量  max\_connect\_errors = 30  # 所有线程所打开表的数量  table\_cache = 4096  # 允许外部文件级别的锁. 打开文件锁会对性能造成负面影响  #external-locking  # 服务所能处理的请求包的最大大小以及服务所能处理的最大的请求大小  max\_allowed\_packet = 32M  # 在一个事务中binlog为了记录SQL状态所持有的cache大小  binlog\_cache\_size = 4M  # 独立的内存表所允许的最大容量.  max\_heap\_table\_size = 128M  # 排序缓冲被用来处理类似ORDER BY以及GROUP BY队列所引起的排序  sort\_buffer\_size = 16M  # 此缓冲被使用来优化全联合(full JOINs 不带索引的联合).  join\_buffer\_size = 16M  # 我们在cache中保留多少线程用于重用  thread\_cache\_size = 16  # 此值只对于支持 thread\_concurrency() 函数的系统有意义( 例如Sun [Solaris](http://www.ha97.com/category/unix/solaris" \o "Solaris)) # 推荐 [CPU数量]\*(2~4) 来作为thread\_concurrency的值 thread\_concurrency = 8  # 查询缓冲常被用来缓冲 SELECT 的结果并且在下一次同样查询的时候不再执行直接返回结果  query\_cache\_size = 128M  # 此设置用来保护查询缓冲,防止一个极大的结果集将其他所有的查询结果都覆盖.  query\_cache\_limit = 4M  # 被全文检索索引的最小的字长.  ft\_min\_word\_len = 8  # 线程使用的堆大小. 此容量的内存在每次连接时被预留.  thread\_stack = 512K |

**附录参考：**

探秘Java虚拟机——内存管理与垃圾回收：<http://sunbean.blog.51cto.com/972509/768034>

JVM参数调：<http://blog.csdn.net/chenyi8888/article/details/4720489>

（总结）MySQL my.cnf参数配置优化详解：<http://www.ha97.com/4110.html>

Tomcat 生产服务器性能优化：<http://www.oschina.net/translate/tomcat-performance-tuning>

# 性能测试模板

## COD系统性能测试报告范例

**1.概述**

**1.1测试对象**

1. COD前置查询接口
2. COD前置确认接口
3. COD消费接口
4. COD前置撤销接口
5. COD前置退款接口

1.2测试环境

|  |  |
| --- | --- |
| 软件名称 | 版本 |
| Red Hat | 2.6.32-279.el6.x86\_64 |
| ORACLE | 11g |
| JDK | jdk1.6.0\_25 |
| TOMCAT | Apache Tomcat/7.0.42（COD前置）  Apache Tomcat/6.0.33(交易) |

|  |  |
| --- | --- |
| **服务器** | |
| CPU | AMD Opteron(TM) Processor 6220 |
| 内存 | 16G |
| 网络连接 | 局域网 |
| IP地址 | 192.168.195.167 |
| 负载端（2台PC服务器） | |
| 操作系统 | Windows 7 企业版64位 |
| 测试工具 | LoadRunner11 |



**1.3测试模型**

根据业务模型同比缩小后作为本次测试模型。

|  |  |  |  |
| --- | --- | --- | --- |
| 系统名称 | 编号 | 交易名称 | 本业务交易占比 |
| COD项目 | 1 | 消费 | 30 |
| 2 | COD前置查询 | 30 |
| 3 | COD前置确认 | 30 |
| 4 | COD前置撤销 | 5 |
| 5 | COD前置退款 | 5 |

**1.4测试指标**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 测试指标  系统名称 | 系统交易处理能力  （TPS：笔/秒） | 平均响应时间  （TRT：秒） | 系统资源使用率 | 交易失败率%  （不包括返回错误的正常交易） | 本次（Q4）全国日交易量:51（万） | 预计明年6-18日交易量300（万） |
| 消费 | 35 | 2 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 76.5 | 450 |
| COD前置查询 | 23 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 51 | 300 |
| COD前置确认 | 23 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 51 | 300 |
| COD前置撤销 | 0.5 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 1 | 1 |
| COD前置退款 | 0.5 | 1 | 各主机CPU与内存使用率低于60%，内存无大幅增长趋势 | <0.1% | 1 | 1 |

说明：预期目标是根据Q4日交易量下进行测试，明年6-18的目标只是个预期的展望、通过测试可以比对与预期的差距。本次检测系统TPS能否达到47，如果达不到，将找到系统TPS稳定的情况下的最大值。

TPS是根据：2、8 原则估算出来的。  20%的时间80%的业务

例如510000\*0.8/（24\*0.2\*3600）=23/S

**1.5测试前的参数调优**

**ulimit：**

core file size (blocks, -c) unlimited

data seg size (kbytes, -d) unlimited

scheduling priority (-e) 0

file size (blocks, -f) unlimited

pending signals (-i) 124397

max locked memory (kbytes, -l) 64

max memory size (kbytes, -m) unlimited

open files (-n) 1048576

pipe size (512 bytes, -p) 8

POSIX message queues (bytes, -q) 819200

real-time priority (-r) 0

stack size (kbytes, -s) unlimited

cpu time (seconds, -t) unlimited

max user processes (-u) 102400000

virtual memory (kbytes, -v) unlimited

file locks (-x) unlimited

**tomcat线程池：**



**2测试方法**

**2.1负载测试**

**测试目的：**容量测试是针对多支交易，在被测系统测试模型下逐步增加并发用户数进行负载测试，获取各系统处理的最优性能。

**测试方法：**

1. 按照并发梯度加压的方法，逐步提升对被测系统的压力；
2. 直到tps不再上升、系统出现容量拐点或瓶颈后结束负载测试，获取其系统最优处理性能；
3. 观察应用服务器与数据库服务器CPU、内存及磁盘的占用情况；

**2.2疲劳强度测试（混合场景）**

**测试目的：**

检测系统长时间在高负载情况下的性能表现。

**测试方法：**

1. 按照并发梯度加压的方法，逐步提升对被测系统的压力；
2. 当系统达到较高的负载时（最高负载的80%左右），不再提升压力，保持此压力较长时间（10个小时）。
3. 观察被测系统CPU、内存及磁盘的占用情况；

**3.测试结果**

总体情况

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 测试项 | 用户数 | 执行时间 | TPS(笔/秒) | 平均响应时间(秒） | 成功数 | 失败数 | 成功率(%) | 备注 |
| 查询接口 | 100 | 15min | 51 | 1.1 | 67645 | 18 | 99 |  |
| 70 | 15min | 61.2 | 1.26 | 59827 | 0 | 100 |  |
| 50 | 15min | 45 | 1.1 | 45345 | 0 | 100 |  |
| 确认接口 | 50 | 15min | 36.4 | 1.4 | 36453 | 0 | 100 |  |
| 30 | 15min | 26.1 | 1.2 | 23882 | 0 | 100 |  |
| 70 | 15min | 50.3 | 2 | 37028 | 0 | 100 |  |
| 100 | 15min | 60 | 1.28 | 80524 | 0 | 100 | 用户再增加的时候，tps骤降 |
| 撤销接口 | 50 | 15min | 41.9 | 1.36 | 37438 | 0 | 100 |  |
| 70 | 15min | 55.2 | 1.23 | 61520 | 0 | 100 |  |
| 100 | 15min | 34.8 | 1.23 | 52052 | 2 | 100 |  |
| 退款接口 | 50 | 15min | 45.4 | 1.1 | 46233 | 0 | 100 |  |
| 70 | 15min | 7 | 1 | 28397 | 0 | 100 | 10分钟的时候tps骤降 |
| 消费接口 | 10 | 15min | 5 | 1.96 | 4603 | 0 | 100 |  |
| 15 | 15min | 4.9 | 3 | 4652 | 0 | 100 |  |
| 混合 | 50 | 7.5小时 | 查询：1.094确认：1.107撤销：1.095退款：1.087消费：2.317 |  | 1010381 | 709 | 99 | 50vuser。消费30%、查询30%、确认30%，退款5%撤销5% |

应用服务器情况：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **测试项** | **用户数** | **cpu%（平均）** | **cpu%（最大）** | **服务器load（最大）** | **FullGC次数** | **Disk（Iowait）** | **SWAP** | **thread blocked** | **jconsole内存图** | **备注** |
| 查询接口 | 50 | 12.8 | 50 | 2.03 | 0 | 0 |  | 无 | 正常 |  |
| 70 | 20.4 | 100 | 5.94 | 0 | 0 | 0 | 无 | 正常 | CPU最大100%。平均负载过高，应用服务器成为瓶颈 |
| 100 | 17.5 | 39 | 3.74 | 0 | 0 |  | 无 | 正常 |  |
| 确认接口 | 50 | 15.6 | 81.7 | 3.09 | 0 | 0 | 0 | 无 | 正常 |  |
| 30 | 10.9 | 96.3 | 0.54 | 0 | 0 | 0 | 无 | 正常 |  |
| 70 | 13.1 | 92 | 1.05 | 0 | 0 | 0 | 无 | 正常 |  |
| 100 | 22.1 | 100 | 3.18 | 0 | 0 | 0 | 无 | 正常 | 事物失败的时候是负载:2.58.比较大 |
| 撤销接口 | 50 | 34.3 | 100 | 6.4 | 0 | 0 | 0 | 无 | 正常 | 事物失败的时候是负载最大 |
| 70 | 10.6 | 45.3 | 4.4 | 0 | 0 | 0 | 无 | 正常 |  |
| 100 | 22.4 | 100 |  | 0 | 0 | 0 | 无 | 正常 |  |
| 退款接口 | 50 | 12.7 | 94.1 | 0.52 | 0 | 0 | 0 | 无 | 正常 |  |
| 70 | 8.9 | 25.3 | 2.69 | 0 | 0 | 0 | 无 | 正常 |  |
| 消费 | 10 | 5.2 | 31.8 | 3.98 | 1 | 0 | 0 | 无 | 正常 |  |
| 15 | 90.3 | 99.9 | 10.98 | 2 | 0 | 0 | 无 | 正常 |  |
| 混合 | 50 | 61.3 | 99.9 | 13.21 | 8 | 0 | 0 | 无 |  |  |

数据库服务器情况：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试项** | **用户数** | **cpu** | **Iowait（最大）** | **问题** |
| 查询接口 | 100 | 23 | 9.34 | 硬解析比较多，建议变量绑定 |
| 确认接口 | 100 | 28.69 | 7.59 | 硬解析比较多，建议变量绑定 |
| 撤销接口 | 100 | 53.47 | 5.61 | 无 |
| 退款接口 | 50 | 37.37 | 4.49 | 无 |
| 消费接口 | 15 | 34.08 | 1.63 | T\_POS\_ORDER走的是全表扫描 |
| 混合 | 50 | 61.4 | 2.99 | 压到7个多小时的时候，后台报错，具体见后面的分析 |

说明:

1、用户加载方式：每分钟加载10个用户

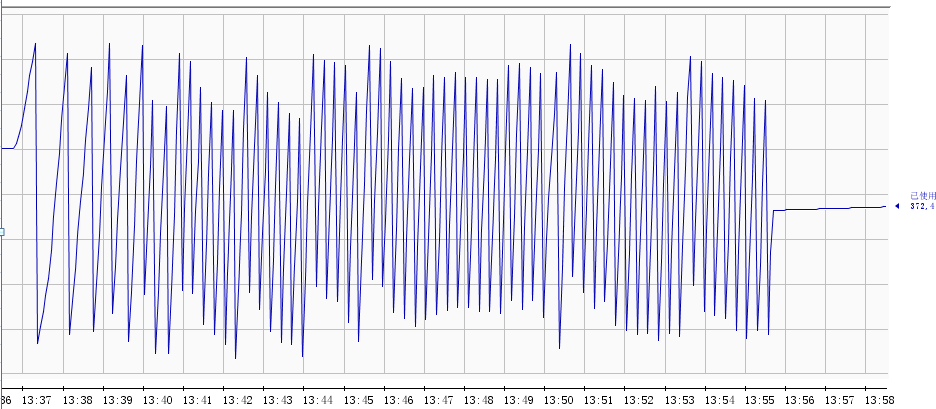
2、使用2台压力机，且压力机负载不高

3、单场景每个接口tps最高的时候。cpu高、负载大，服务瓶颈。

**4.测试结果分析**

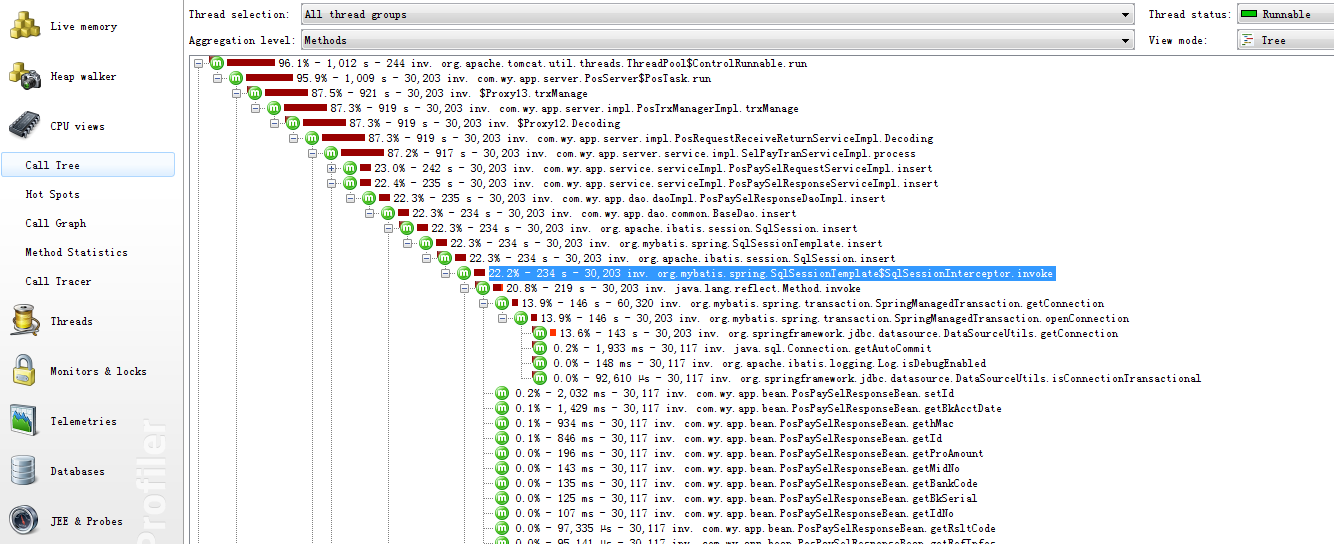
**4.1查询接口**

**jconsole图：**



**结论：JVM回收彻底，无Full GC。内存正常。**

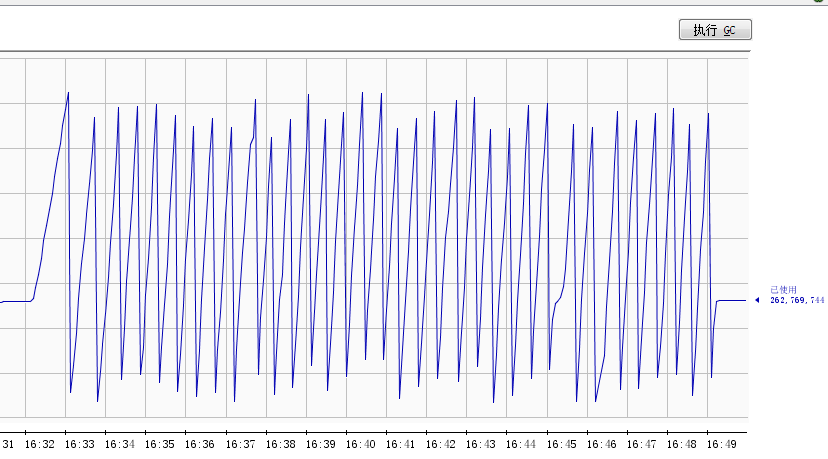
**jprofiler图（最消耗cpu方法占比）：**



分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding。开发进行调优的时候可以参考此图。

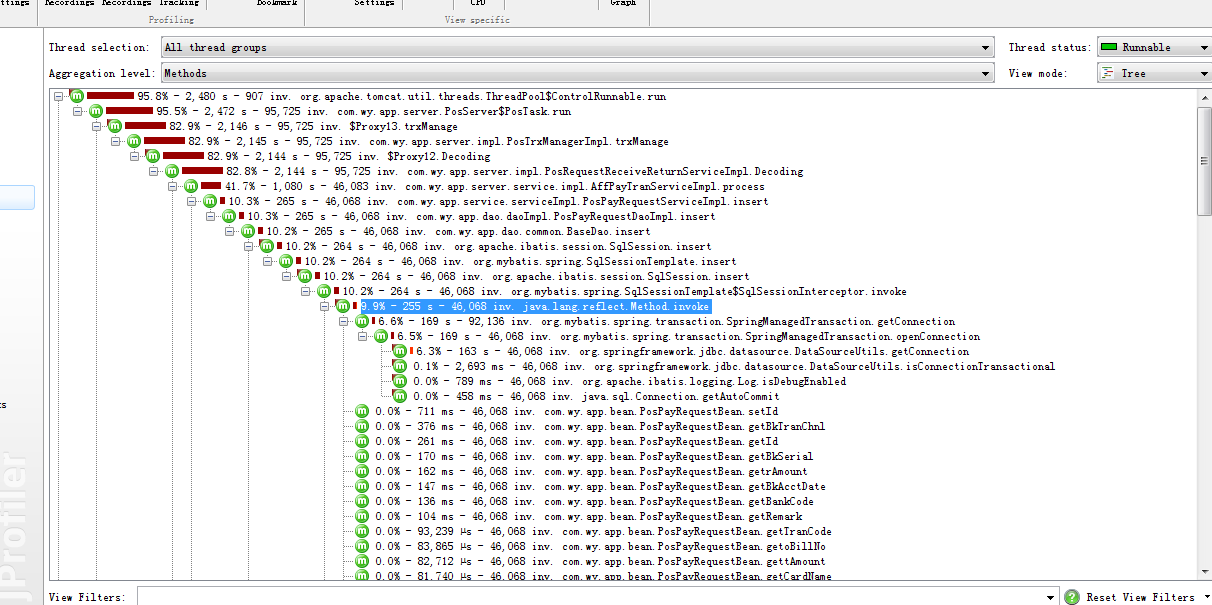
**4.2确认接口**

**jconsole图：**



**结论：JVM回收彻底，无Full GC。内存正常。**

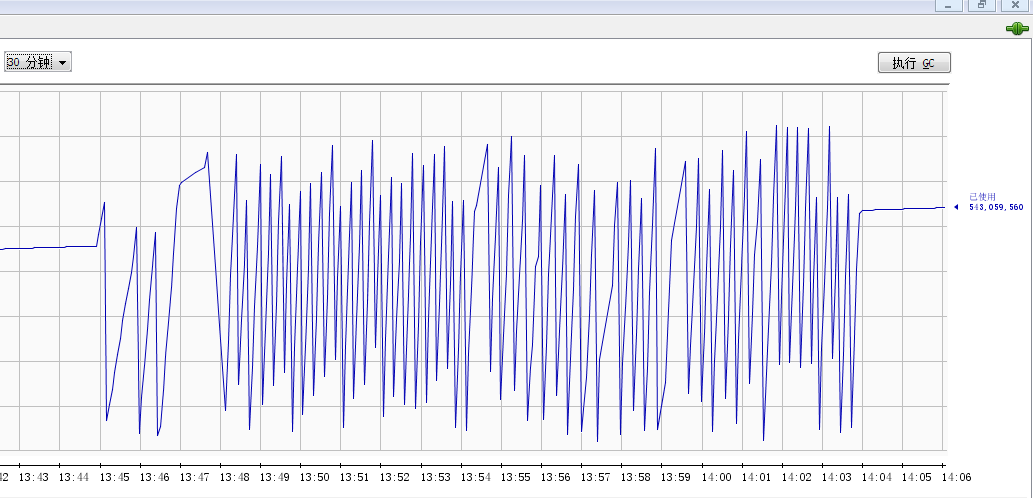
**jprofiler图（最消耗cpu方法占比）：**



分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding、com.wy.app.server.service.impl.AffPayTranServiceImpl.process。开发进行调优的时候可以参考此图。

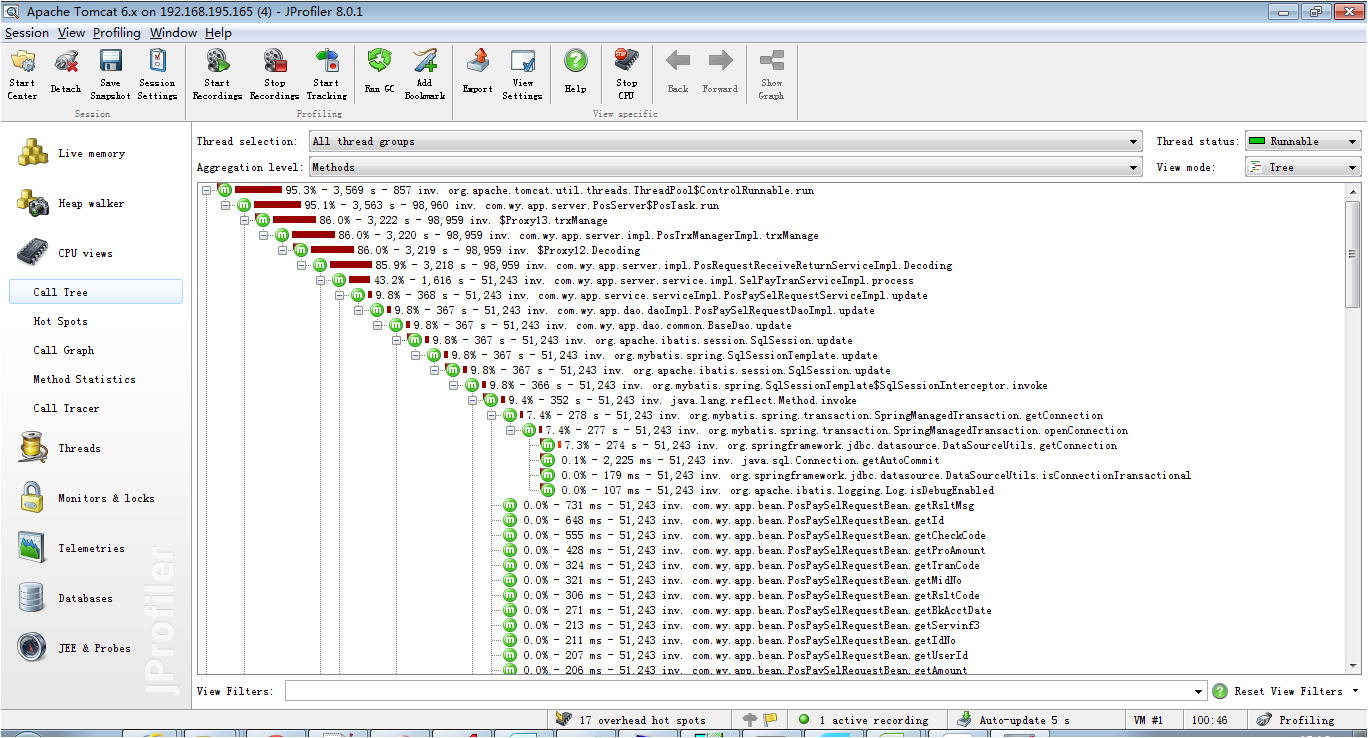
**4.3撤销接口**

**jconsole图：**



**结论：JVM回收彻底，无Full GC。内存正常。**

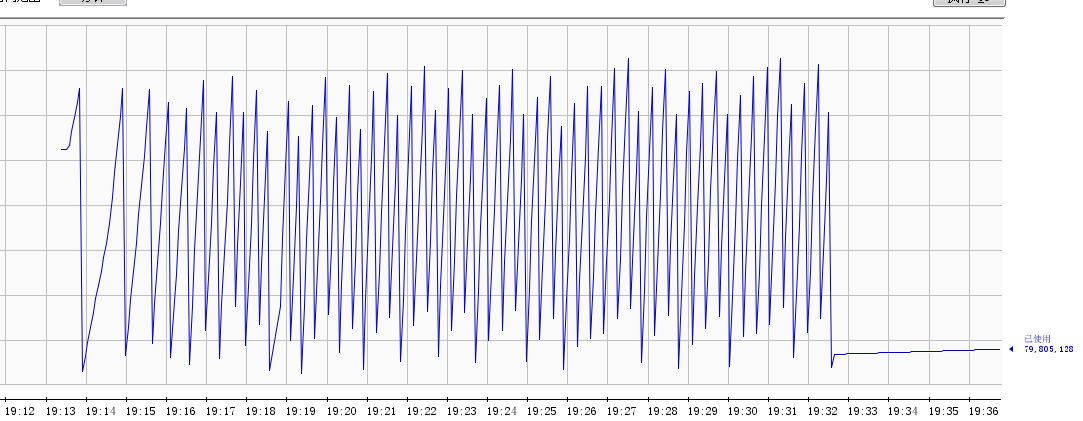
**jprofiler图（最消耗cpu方法占比）：**



分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding、com.wy.app.server.service.impl.SelPayTranServiceImpl.process开发进行调优的时候可以参考此图。

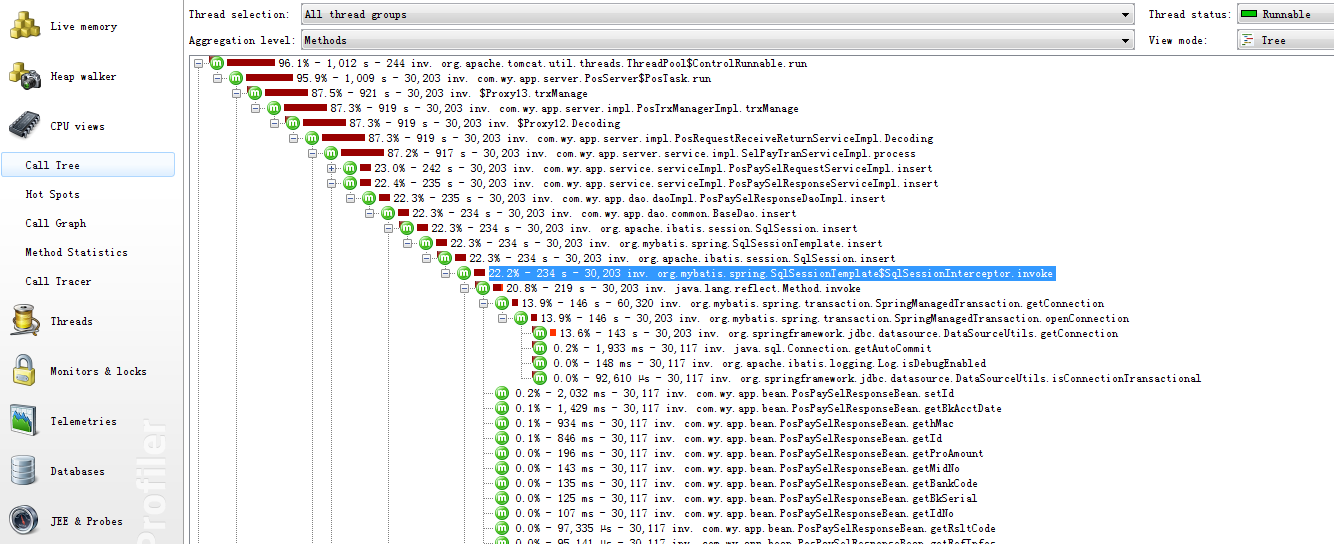
**4.4退款接口**

**jconsole图：**



**结论：JVM回收彻底，无Full GC。内存正常。**

**jprofiler图（最消耗cpu方法占比）：**



分析：cpu占比比较高的方法为：com.wy.app.server.PosServer..PosTask.run、com.wy.app.server.impl.PosTrxManagerImpl.trxManage、com.wy.app.server.impl.PosResquestReceiveReturnServiceImpl.Decoding、com.wy.app.server.service.impl.BackPayTranServiceImpl.process、com.wy.app.server.service.impl.BackPayTranServiceImpl.processPayBack。开发进行调优的时候可以参考此图。

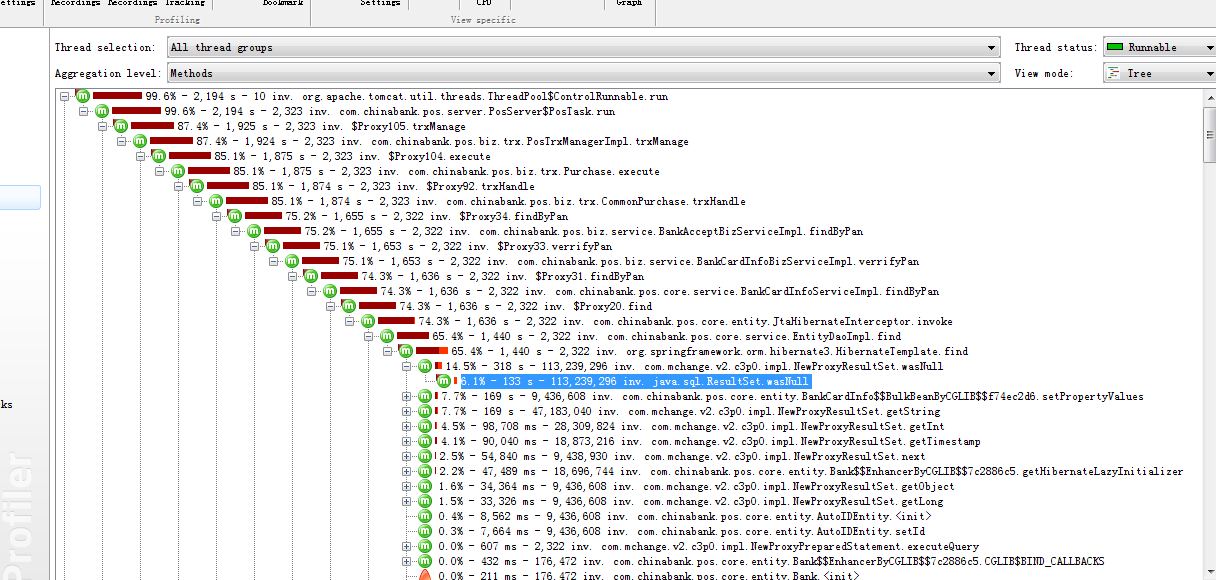
**4.5消费接口**

**jconsole图：**



**结论：JVM回收彻底，无Full GC。内存正常。**

**jprofiler图（最消耗cpu方法占比）：**

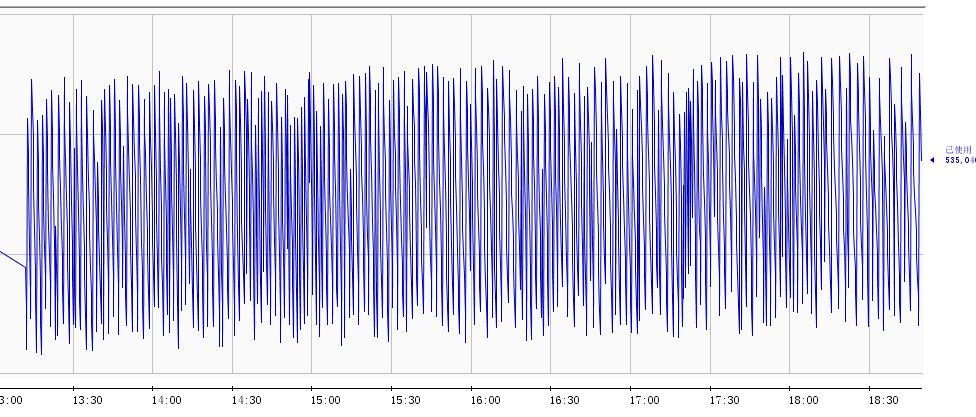


分析：cpu占比比较高的方法为：com.chinabank.pos.server.PosSERVER..PosTask.run、com.chinabank.pos.biz.trx.PosTrxManagerImpl.trxManage、com.chinabank.pos.biz.trx.Purchase.execute、com.chinabank.pos.biz.trx.CommonPurchase.trxHandle、com.chinabank.pos.biz.service.BankAcceptBizServiceImpl.findByPan。开发进行调优的时候可以参考此图。

**4.6混合场景接口**

**jconsole图：**

**COD:**



**消费：**



**结论：cod前置接口：JVM回收彻底，无Full GC。内存正常。**

**消费接口：JVM回收彻底，8次Full GC，属于高负载，1个小时左右一次Full GC。**

**后台问题：在压到7个多小时的时候后台报错：**

### Error updating database. Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Could not get JDBC Connection; nested exception is java.sql.SQLException: Listener refused the connection with the following error:

ORA-12516, TNS:listener could not find available handler with matching protocol stack

The Connection descriptor used by the client was:

192.168.201.41:1521/payment

### Cause: org.springframework.jdbc.CannotGetJdbcConnectionException: Could not get JDBC Connection; nested exception is java.sql.SQLException: Listener refused the connection with the following error:

ORA-12516, TNS:listener could not find available handler with matching protocol stack

The Connection descriptor used by the client was:

192.168.201.41:1521/payment

atorg.mybatis.spring.MyBatisExceptionTranslator.translateExceptionIfPossible(MyBatisExceptionTranslator.java:73)

at org.mybatis.spring.SqlSessionTemplate$SqlSessionInterceptor.invoke(SqlSessionTemplate.java:365)

at $Proxy7.update(Unknown Source)

at org.mybatis.spring.SqlSessionTemplate.update(SqlSessionTemplate.java:251)

at com.wy.app.dao.common.BaseDao.update(BaseDao.java:49)

at com.wy.app.dao.daoImpl.PosPaySelRequestDaoImpl.update(PosPaySelRequestDaoImpl.java:22)

at com.wy.app.service.serviceImpl.PosPaySelRequestServiceImpl.update(PosPaySelRequestServiceImpl.java:25)

at com.wy.app.server.service.impl.SelPayTranServiceImpl.process(SelPayTranServiceImpl.java:138)

at com.wy.app.server.impl.PosRequestReceiveReturnServiceImpl.Decoding(PosRequestReceiveReturnServiceImpl.java:49)

at sun.reflect.GeneratedMethodAccessor92.invoke(Unknown Source)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)

at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:198)

at $Proxy12.Decoding(Unknown Source)

at com.wy.app.server.impl.PosTrxManagerImpl.trxManage(PosTrxManagerImpl.java:18)

at sun.reflect.GeneratedMethodAccessor89.invoke(Unknown Source)

at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)

at java.lang.reflect.Method.invoke(Method.java:597)

at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)

at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:198)

at $Proxy13.trxManage(Unknown Source)

at com.wy.app.server.PosServer$PosTask.run(PosServer.java:85)

at org.apache.tomcat.util.threads.ThreadPool$ControlRunnable.run(ThreadPool.java:687)

at java.lang.Thread.run(Thread.java:662)

Caused by: org.apache.ibatis.exceptions.PersistenceException:

报该错后，tps急剧下降，最终为0，但服务没有宕掉。

**5.测试结论**

1、查询接口支持的最大tps：61.2。响应时间：1.26。支持最大虚拟并发用户数：70。

确认接口支持的最大tps：60。响应时间：1.28。支持最大虚拟用户数：100。

撤销接口支持的最大tps：55.2。响应时间：1.23。支持最大虚拟用户数：70。

退款接口支持的最大tps：45.4。响应时间：1.1。支持最大虚拟用户数：50。

消费接口支持的最大tps：5。响应时间：1.96。支持最大虚拟用户数：10。

2、COD前置查询、确认、撤销、退款接口测试能达到Q4，距明年618相差甚远。但本次依据的是Q4,这4个接口---通过；消费tps（5）达不到预期目标（35）---不通过；

3、混合场景在稳定性测试，报数据库连接异常（COD部分）具体报错见4.6。是否因为没使用数据库连接池所导致？---混合场景测试不通过。

4、建议优化

a、T\_POS\_ORDER表走的是全表扫描，建议加索引。

b、测试查询、确认接口的时候，数据库走的几乎是硬解析，建议变量绑定。

c、消耗CPU较高的方法也建议优化，如4。

## 性能测试报告模板

模板请从wiki下载。

# 性能测试工具

## ****JMeter参考手册****

### ****JMeter介绍****

**JMeter简介**

Apache JMeter 是Apache组织的开放源代码项目，是一个100%纯Java桌面应用，用于压力测试和性能测量。它最初被设计用于Web应用测试但后来扩展到其它测试领域。

Apache JMeter可以用于对静态的和动态的资源（文件，Servlet，Perl脚本，Java对象，数据库和查询，FTP服务器或是其它资源）的性能进行测试。JMeter可以用于在服务器，网络或对象模拟繁重的负载来测试它们提供服务的受压能力或者分析不同压力条件下的总体性能情况。你可以使用JMeter提供的图形化界面，分析性能指标或者在高负载情况下测试你的服务器/脚本/对象。

**JMeter特点**

a、能够对HTTP和FTP服务器进行压力和性能测试，也可以对任何数据库进行同样的测试（通过JDBC）。

b、完全的可移植性和100%纯Java。

c、完全Swing的轻量级组件支持（预编译的JAR使用 javax.swing.\*）包。

d、完全多线程框架允许通过多个线程并发取样和通过单独的线程组对不同的功能同时取样。

e、精心的GUI设计允许快速操作和更精确的计时。

f、缓存和离线分析/回放测试结果。

g、高可扩展性：

     1)可链接的取样器允许无限制的测试能力。

     2)各种负载统计表和可链接的计时器可供选择。

     3)数据分析和可视化插件提供了很好的可扩展性以及个性化。

     4)具有提供动态输入到测试的功能（包括JavaScript）。

     5)支持脚本变成的取样器（在1.9.2及以上版本支持BeanShell）。

**JMeter历史及未来**

JMeter最早是由Apache软件项目的开发工程师Stefano Mazzocchi 开发出来的。他当初开发JMeter主要用于测试Apache JServ(一个被Apache Tomcat 工程取代了的项目)的性能。之后我们又再次设计了JMeter，增加了图形化界面和一些功能测试能力。

随着开发人员借助它的可嵌入的结构，我们希望看到JMeter的能力也随之快速的被扩大。更进一步的主要发展目标是把JMeter在没有减弱它的负载测试的能力的同时尽可能的做成最有效的回归测试工具。

### ****JMeter安装****

**JMeter安装需求**

Java版本：JVM1.4或更高

操作系统：Windows、Linux等

**JMeter安装**

下载：<http://jmeter.apache.org/> 下载最新版

安装：解压即可（避免在一个有空格的路径安装 JMeter，可能导致远程测试出现问题）

### JMeter运行

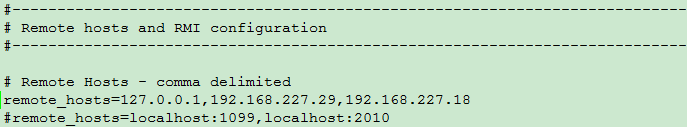
运行 jmeter.bat (for Windows) 或者 jmeter (for Unix) 文件，文件在bin目录下

#### 命令行模式运行

示例：jmeter -n -t test.jmx -l test.jtl

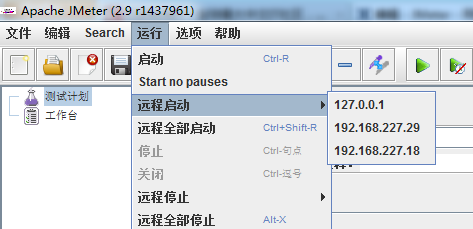
#### 集群模式运行

第一步：jmeter客户端打开jmeter.properties文件，修改remote\_hosts参数，添加jmeter服务端IP，如下图：



第二步：启动jmeter服务端——运行jmeter-server（Linux）或jmeter-server.bat（Windows）

第三部：启动jmeter客户端，选择“运行”->“远程启动”，如下图：



#### 命令行选项

-h, --help 打印使用信息并退出

-v, --version 打印版本信息并推出

-p, --propfile {argument} 使用的JMeter属性文件

-q, --addprop {argument} 附加的属性文件

-t, --testfile {argument} 运行的JMeter测试文件(.jmx)

-l, --logfile {argument} 日志取样文件

-n, --nongui 非用户界面运行JMeter

-s, --server 运行JMeter服务器

-H, --proxyHost {argument} 设置JMeter使用的代理服务器

-P, --proxyPort {argument} 设置JMeter使用的代理服务器端口

-u, --username {argument} 设置JMeter使用的代理服务器用户名

-a, --password {argument} 设置JMeter使用的代理服务器密码

-J, --jmeterproperty {argument}={value} 定义附加的 JMeter 属性

-D, --systemproperty {argument}={value} 定义附加的 System 属性

-S, --systemPropertyFile {filename} 一个属性文件被做为系统属性添加

-L, --loglevel {argument}={value} 定义日志等级: [category=]level，例如 jorphan=INFO or jmeter.util=DEBUG

-r, --runremote从非用户界面模式启动远程服务器

-d, --homedir {argument} 使用的JMeter目录

### ****JMeter常用协议****

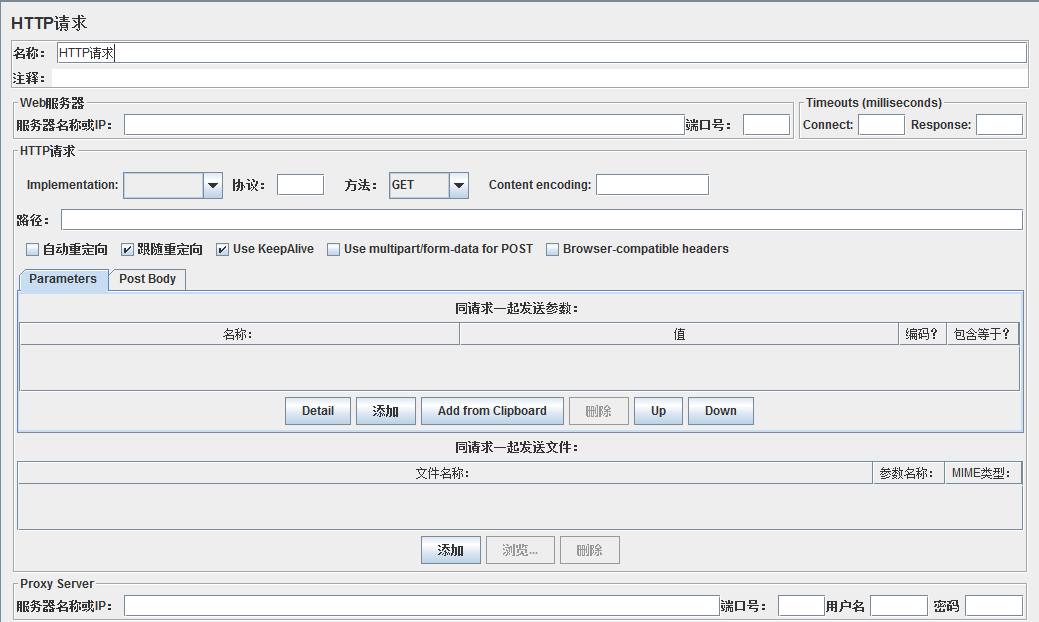
#### HTTP请求

**录制http请求**

使用Badboy（<http://www.badboy.com.au/> 下载）录制后，选择“File”->“Export to JMeter”，导出jmx文件

**编辑http请求**

打开Badboy导出的jmx文件并选择要编辑的HTTP请求Sampler或新建一个HTTP请求Sampler，右侧为编辑区域，如下图：



#### https请求

选择“选项”->“SSL管理器”，加载证书文件，如下图：



#### Java请求

**编写测试类**

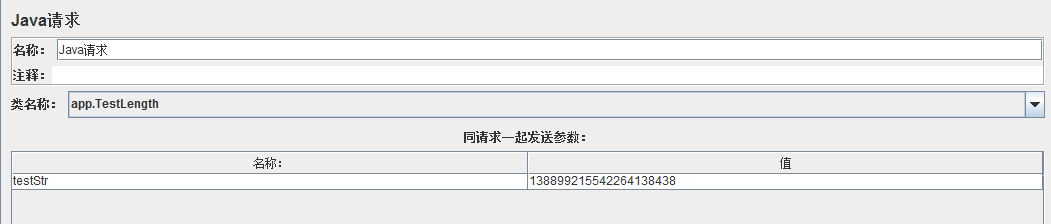
Eclipse创建项目，编写测试类，并将工程导出为jar文件（工程需要依赖 JMETER\_HOME/lib/ext下的ApacheJMeter\_core.jar和ApacheJMeter\_java.jar两个包）

以下示例为一个测试类，该测试类判断输入的字符串长度，如果大于等于5则返回成功，否则返回失败

|  |
| --- |
| importjava.util.Random;  importorg.apache.jmeter.config.Arguments;  importorg.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;  importorg.apache.jmeter.protocol.java.sampler.JavaSamplerContext;  importorg.apache.jmeter.samplers.SampleResult;    publicclassTestLength extendsAbstractJavaSamplerClient {      privateSampleResult results;      privateRandom r = newRandom();      privateString testStr;      publicvoidsetupTest(JavaSamplerContext context) {          testStr = context.getParameter("testStr", "");      }      publicArguments getDefaultParameters() {          Arguments args = newArguments();          args.addArgument("testStr", "abc");          returnargs;      }      @SuppressWarnings("deprecation")      publicSampleResult runTest(JavaSamplerContext context) {          intlen = 0;          results = newSampleResult();          results.sampleStart();          testStr = context.getParameter("testStr", System.currentTimeMillis()                  + ""+ r.nextInt(100000000));          len = testStr.length();          results.sampleEnd();          results.setDataEncoding("UTF-8");          if(len < 5) {              System.out.println(testStr);              results.setResponseCode("testStr:"+ testStr);              results.setResponseData("testStr:"+ testStr);              results.setSuccessful(false);          } else{              System.out.println(testStr);              results.setResponseCode("testStr:"+ testStr);              results.setResponseData("testStr:"+ testStr);              results.setSuccessful(true);          }          returnresults;      }      publicvoidteardownTest(JavaSamplerContext arg0) {      }  } |

**JMeter加载测试类**

将导出的jar文件放到JMETER\_HOME/lib/ext下（测试类依赖的jar包也要放到JMETER\_HOME/lib/ext下），如下图：

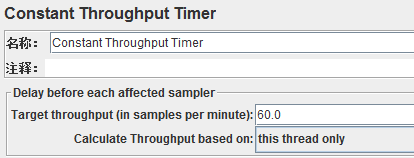


### ****JMeter常用场景****

#### 稳定TPS测试

测试需求为待测系统满足一定TPS要求，则需JMeter模拟稳定的TPS压力来测试，具体方法为：

右键取样器，选择“定时器”->“Constant Throughput Timer”，如下图：

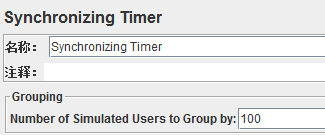


其中，“Target throughput”是设置的固定TPS值，单位是：个/分钟

#### 并发压力测试——设置集合点

通常来说，JMeter的线程数即为并发的压力数，实际上JMeter在运行时，每个线程是独立的，虽然有100个线程，但这些线程并不是同时向服务器发送请求，JMeter要模拟100个用户在同一时间点向服务器发送请求，才是真正的并发压力，具体方法为：

右键取样器，选择“定时器”->“Synchronizing Timer”，如下图：

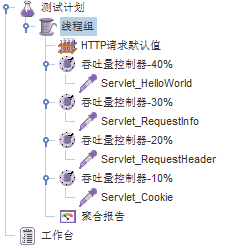


Synchronizing Timer中设置的压力数小于等于线程组中的并发数

#### 混合压力场景测试

稳定性测试中，经常需要多个接口按一定比例分配来模拟真实的场景，称为混合压力场景测试，JMeter通过吞吐量控制器实现，具体方法为：

在取样器的上一级添加“逻辑控制器”->“吞吐量控制器”，如下图：



“吞吐量控制器”的比例计算公式如下：

1、Without Per User——Percent Executions：执行次数=（线程数\*循环次数）\*吞吐量%

2、Without Per User——Total Executions：执行次数=MIN（吞吐量，线程数\*循环次数）

3、With Per User——Percent Executions：执行次数=线程数\*（循环次数\*吞吐量%）

注：（循环次数\*吞吐量%）2.5时为2,2.55时为3

4、With Per User——Total Executions：执行次数=MIN（线程数\*吞吐量，线程数\*循环次数）

#### 参数化

**自定义的变量**

方法：配置元件->用户定义的变量

格式：${Parameter\_Name}

**用JMeter中的函数获取参数值**

方法：打开函数助手

格式：${\_\_Random(0,100,)}



**从文件中读取**

方法：配置元件->CSV Data Set Config

Allow quoted data：允许双引号内的数据作为参数，逗号分隔

Recycle on EOF：循环读文件

Stop thread on EOF：当不循环读文件时，结束线程，不结束则读出EOF

**用正则表达式从前面请求的响应数据中提取**

说明：JMeter中包含范本匹配软件Apache Jakarta ORO

方法：后置处理器->正则表达式处理器

()：封装了待返回的字符串

.：匹配任何字符

+：一次或多次

?：找到第一个匹配项后停止

[^"]：匹配任何字符除了"

([^"]+)优于(.+?)：因为前者匹配的引擎在找到第一个右侧"后，就会停止搜索，而后者的匹配引擎会去寻找">

正则表达式模板：$1$、$2$，例如：

引用名称：MYREF

正则表达式：name="(.+?)" value="(.+?)"

模板：$1$$2$

MYREF: file.namereadme.txt

MYREF\_g0: name="file.name" value="readme.txt"

MYREF\_g1: file.name

MYREF\_g2: readme.txt

**从数据库中获取**

方法：创建JDBC Request从数据库中获取数据

SQL：select concat ("param\_value:","'",param\_value,"'") from parameters where param\_name='Name';输出为：

param\_value:'YangFei'

**用BeanShell Sampler生成参数**

方法：Java类导出成jar到%JMETER\_HOME/lib/ext下

格式：vars.put("param\_name",param\_value);

### ****JMete主要组件介绍****

详细文档参见：<http://jmeter.apache.org/usermanual/component_reference.html>

#### 线程组

线程组元件是任何一个测试计划的开始点。在一个测试计划中的所有元件都必须在某个线程组下。顾名思义，线程组元件控制JMeter执行你的测试计划时候使用的线程数量。

对线程组的控制允许你：

1. 设置线程数
2. 设置ramp-up period
3. 设置测试要执行的次数

每个线程将会完全独立的运行测试计划，互不干扰。多个线程用于模仿对服务器的并发访问。

ramp-up period指示JMeter用于达到全部选择的线程的时间。如果选择了10个线程，并且ramp-up period是100秒，那么JMeter将使用100秒使10个线程启动并运行。每个线程将在前一个线程启动后10（100/10）秒后启动。如果有 30个线程并且ramp-up period是120秒，那么线程间隔4秒。

Ramp-up需要足够长的时间来避免在开始测试时产生太大的负荷，并且有足够短的时间使最后一个线程在第一个线程结束之前运行。

开始的Ramp-up = 线程数+必须的调节数

缺省情况下，线程组被配置为不确定的循环执行它下面的元件。另外，你可以设置线程组在结束前循环的次数。如果次数设置为1，那么JMeter在停止前只执行测试计划一次。

#### 控制器

JMeter有两种控制器：取样器和逻辑控制器。

取样器告诉JMeter发送请求到服务器。例如，如果你要JMeter发送一个HTTP请求则添加一个HTTP请求取样器。你也可以通过添加一个或者多个配置元件到一个取样器来定制一个请求。

逻辑控制器让你定制当发送请求时JMeter使用的判断逻辑。例如，你可以添加交替控制器来在两个HTTP请求取样器之间交替。

#### 取样器

JMeter取样器包括：

* FTP 请求
* HTTP 请求
* JDBC 请求
* Java 请求
* LDAP 请求
* SOAP/XML-RPC 请求
* WebService (SOAP) 请求

每个取样器有一些你可以设置的属性。你可以通过添加一个或多个配置元件到取样器来进一步定制它。注意JMeter发送请求按照取样器出现在树中的顺序。

如果你想发送多个相同类型的请求（例如，HTTP Request）到相同的服务器，可以考虑使用一个默认配置元件。每个控制器有一个或者多个默认配置元件（见下）。

记得添加一个监听器到线程组来查看/保存你的请求结果到磁盘。

如果你对使用JMeter平台的基础验证器到你的请求响应感兴趣，添加一个断言到请求控制器。例如，在压力测试一个web程序时，服务器会返回一个成功的HTTP响应代码，但是这个页面有错误或者被忽略部分。你可以添加断言来检查某个HTML标签，一些错误字符串，等等。JMeter允许你使用正则表达式创建断言。

#### 逻辑控制器

逻辑控制器让你定制当发送请求时JMeter使用的判断逻辑。逻辑控制器还可以作为下列任何元件的子元件：取样器（请求）、配置元件、和其他逻辑控制器。逻辑控制器可以改变来自它们的子元件的请求顺序。它们可以修改请求本身，导致JMeter重复请求，等。

理解逻辑控制器在测试计划中的效果，考虑下列测试树：

* 测试计划
  + 线程组
    - 仅一次控制器
      * 登录请求(一个HTTP请求)
    - 加载搜索页面(HTTP取样器)
    - Interleave Controller
      * 搜索"A"(HTTP取样器)
      * 搜索"B"(HTTP取样器)
      * HTTP默认请求(配置元件)
    - HTTP默认请求(配置元件)
    - Cookie管理器(配置元件)

这个测试的第一件事就是登录请求仅在第一次经过时被执行。随后的迭代会忽略它。这应使用仅一次控制器。

登陆后，下一个取样器加载搜索页面（假设一个用登录的web应用程序，并到达搜索页面去搜索）。这仅是一个简单的请求，不会被任何逻辑控制器过滤。

加载搜索页面后，我们要做一个搜索。事实上，我们想做两个不同的搜索。然而，在每个搜索之间我们想要自己重新加载搜索页面。我们通过4个简单 HTTP元件这样做。

注意HTTP默认请求属于插入控制器。假如"Search A"和"Search B"共享同样的PATH信息（一个HTTP请求说明中包括域，端口，方法，协议路径和参数，附加其他可选项）。两个搜索请求访问同样的后端搜索引擎（比方说Servle，或者cgi脚本），这样是说得通的。与其两者都配置使用相同信息的HTTP取样器，我们可以抽象那些新到一个单独的配置元件。当内部控制器通过"Search A"或者"Search B"传递时，它会从HTTP默认请求配置元件中获得值填充空白。所以我们可以为那些请求保留PATH域为空，然后把那些信息放到配置元件。在这个例子中，这至多是一个很小的好处，但它显示了这个特性。

在这个树中下一个元件是另一个HTTP默认请求，这个时间被添加到线程组本身。这个线程组有一个内建的逻辑控制器，因此它正好使用这个配置元件做为上面的描述。它填充任何穿过的请求的空白。在web程序中你所有的HTTP取样器元件DOMAIN域为空，这是极度有用的，替代的，把那些信息放到HTTP默认请求元件中，添加到线程组。通过这样做，你可以在一个同的服务器通过改变你测试计划中的一个域来测试你的程序。另外，你必须编辑每个取样器。

最后一个元件是一个HTTP Cookie管理器。一个Cookie管理器应该添加到所有web测试上-否则JMeter会忽略Cookie。通过在线程组级添加它，我们可以确定所有的线程分享同样的Cookie。

逻辑控制器可以组合达到不同的结果。

#### 监听器

监听器提供访问JMeter收集当JMeter运行的关于测试计划的信息。图形结果监听器在一张图上绘制响应时间。"查看结果树"监听器显示了请求和响应取样器的细节，并且以基础的HTML和XML显示响应表现。其他监听器提供了摘要或者集合信息。

另外，监听器可以指导它们收集的数据到一个文件供以后用。在JMeter中每一个监听器提供一个域来指出存储数据的文件。

在测试中监听器可以添加到任何位置。它们仅仅会从它们等级或者它们以下等级的元件收集数据。

#### 定时器

默认，JMeter线程发送请求时不在请求间暂停。我们建议你通过添加一个可用的定时器到你的线程组来指定一个延迟。如果你不添加延迟，JMeter会在短时间内产生太多请求，可能会压倒你的服务。

定时器会使JMeter在一个线程开始每个请求间延迟一段时间。

如果你选择添加多于一个定时器到一个线程组，JMeter会在执行取样器前获得定时器数量并暂停那个时间量。

#### 断言

断言允许你断言关于从测试服务器收到的响应的行为。使用断言你本质上你可以测试你的应用程序返回你期望的结果。

例如，你可以断言一个查询的响应会包含一些特殊的文本。你指定的文本可能是Perl风格的正则表达式，并且你可以指出这个响应是包含这个文本，还是匹配整个响应。

你可以添加一个断言到任何取样器。例如你可以添加一个断言到HTTP请求检查文本"</HTML>"。JMeter会检查在HTTP响应中表现的文本。如果JMeter没有找到这个文本，它会标记这个为一个失败的请求。

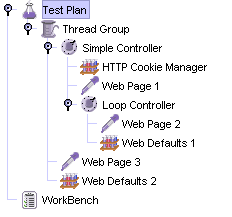
为了查看断言结果，添加一个断言监听器到线程组。

#### 配置元件

配置元件配合取样器工作。虽然它不发送请求（除了HTTP代理服务器），但是它可以添加或者修改请求。

一个配置元件进能访问有所代替元件所在的树分支的内部。例如，如果你在一个简单逻辑控制器里面设置一个HTTP Cookie管理器，Cookie管理器很容易访问"web Page 1"和"web Page 2"HTTP请求。但是不能访问"web Page 3"。

同样，一个在树枝内部的配置元件比在父支的同样元件有更高的优先级。例如，我们定义两个HTTP默认请求元件，"Web Defaults 1"和"Web Defaults 2"。如果我们把"Web Defaults 1"放置在一个循环控制器内部，仅"Web Page 2"可以访问它。另一HTTP请求会使用"Web Defaults 2"，如果我们把它放置在线程组 (所有其他分支的父支)。



#### 前置处理器

前置处理器在取样器请求建立前执行一些行为。如果前置处理器附属于取样器元件，那么它会仅在那个取样器元件运行前执行。前置处理器最常用来在取样请求运行前修改它的设置，或者更新不能从响应文本提取的变量。

#### 后置处理器

后置控制器在取样器请求建立后执行一些行为。如果后置处理器附属于取样器元件，那么它会仅在那个取样器元件运行后执行。后置处理器最常用来处理响应数据，常用来从它里面提取数值。

#### 执行顺序

1. 配置元件
2. 前置处理器
3. 定时器
4. 取样器
5. 后置处理器
6. 断言
7. 监听器

### ****JMeter函数****

详细文档参见：<http://jmeter.apache.org/usermanual/functions.html>

#### 函数列表

* regexFunction - 正则表达式计数器
* counter
* threadNum - 得到线程数
* intSum - 添加变量
* StringFromFile - 从文件读取一行
* machineName - 得到本地计算机名
* JavaScript (Apache Rhino)
* random number
* CSVRead - 从CSV文件读取
* 读取一个属性
* P - 读取一个属性
* setProperty - 设置一个属性
* log - 记录一个日志
* logn - 记录一个日志
* BeanShell - 运行BeanShell
* split - 分隔一个字符串为变量
* XPath - 使用一个xpath表达式
* time - 返回一些格式的当前时间
* jexl - 执行一个jexl表达式

### ****JMeter扩展****

#### JMeter监控统计插件

详见：<http://jmeter-plugins.org/>

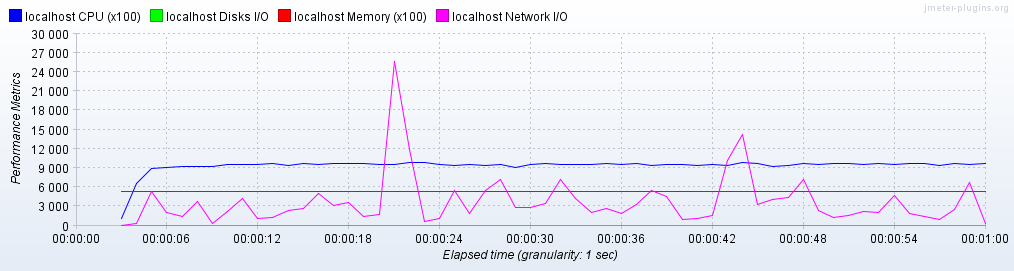
下载地址：<http://jmeter-plugins.org/downloads/all/>

常用监控统计视图如下：

**PerfMon Metrics Collector and Server Agent**

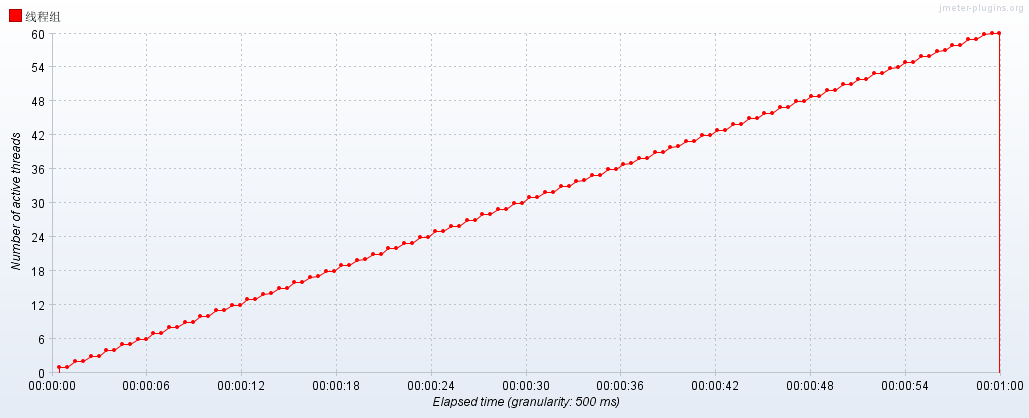
JMeter通过PerfMon Metrics Collector视图远程监控服务器资源，如：CPU、Memory、Disk I/O、Network I/O等

注：需要在服务器端启动Server Agent，默认端口为4444



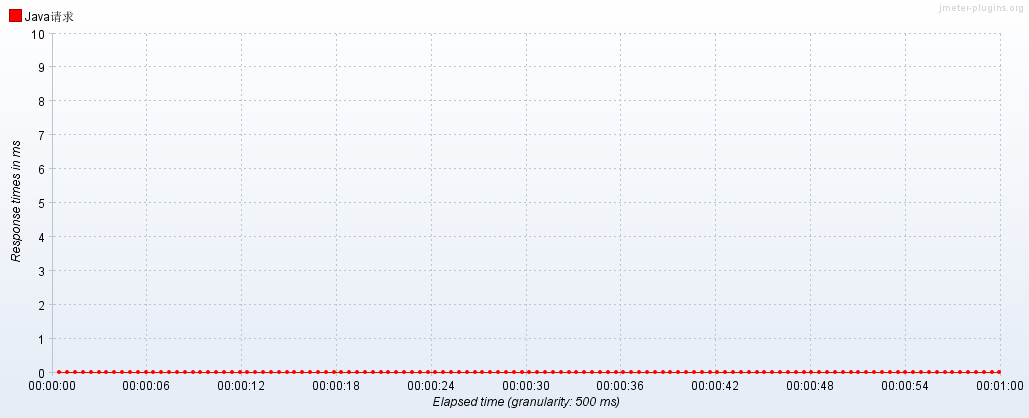
**Active Threads Over Time**

JMeter通过Active Threads Over Time视图监控活动的线程数



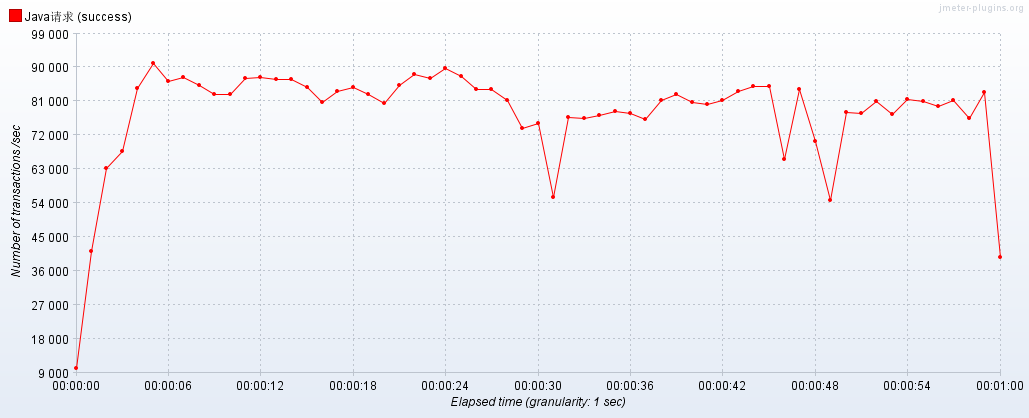
**Response Times Over Time**

JMeter通过Response Times Over Time视图监控响应时间



**Transactions per Second**

JMeter通过Transactions per Second视图监控TPS



### JMeter持续集成

#### JMeter+Ant

JMeter 的 extras 目录包含一个示例 build.xml 脚本，说明了 JMeter Ant 任务的用法

ant-jmeter-1.1.1.jar下载地址：<http://www.programmerplanet.org/media/ant-jmeter/ant-jmeter-1.1.1.jar>

第一步：运行JMeter并创建一个测试计划

第二步：在 Ant 中定义 JMeter 任务

|  |
| --- |
| <taskdefname="jmeter"classname="org.programmerplanet.ant.taskdefs.jmeter.JMeterTask"/> |

第三步：从 Ant 运行 JMeter

|  |
| --- |
| <jmeter  　　jmeterhome="${jmeter.home}"  　　resultlog="${basedir}/target/JMeterResults.xml">  　　<testplansdir="${basedir}/tests/load"includes="BreweryTestPlan.jmx"/>  　　</jmeter> |

第四步：使用 XSLT 创建 JMeter HTML 报告

|  |
| --- |
| <xsltin="${basedir}/target/JMeterResults.xml"  　　out="${basedir}/target/JMeterResults.html"  　　style="${jmeter.home}/extras/jmeter-results-detail-report\_21.xsl"/> |

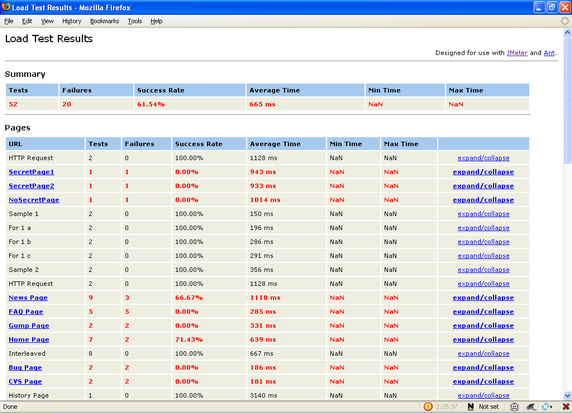
第五步：向 JMeter 传递可选参数和属性

|  |
| --- |
| <jmeter  　　jmeterhome="${jmeter.home}"  　　resultlog="${basedir}/target/JMeterResults.xml">  　　<jvmargvalue="-Xincgc"/>  　　<jvmargvalue="-Xmx128m"/>  　　<jvmargvalue="-Dproperty=value"/>  　　<propertyname="request.threads"value="5"/>  　　<propertyname="request.loop"value="50"/>  　　<propertyname="jmeter.save.saveservice.assertion\_results"value="all"/>  　　<propertyname="jmeter.save.saveservice.output\_format"value="xml"/>  　　<testplansdir="${basedir}/tests/load"includes="BreweryTestPlan.jmx"/>  　　</jmeter> |

第六步：定时运行任务

|  |
| --- |
| ...  　　<modificationset>  　　<svnRepositoryLocation="${svnrepo.location}"/>  　　<timebuildusername="admin"time="2300"/>  　　</modificationset>  　　... |

第七步：生成JMeter HTML报告，如下图：



#### JMeter+Maven

maven-jmeter-plugin地址：<https://github.com/afranken/jmeter-maven-plugin>

第一步：创建一个src/test/jmeter目录，把JMeter测试文件放这

第二步：添加plugin到pom.xml

|  |
| --- |
| <plugin>      <groupId>com.lazerycode.jmeter</groupId>      <artifactId>jmeter-maven-plugin</artifactId>      <version>1.8.1</version>      <executions>          <execution>              <id>jmeter-tests</id>              <phase>verify</phase>              <goals>                  <goal>jmeter</goal>              </goals>         </execution>      </executions>  </plugin> |

第三步：运行mvn verify

#### JMeter+Jenkins

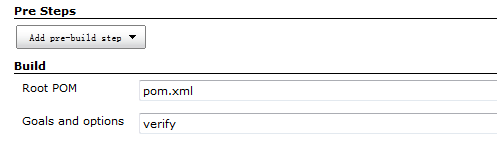
Performance Plugin下载地址：<https://github.com/hudson3-plugins/performance-plugin>

第一步：在Jenkins中安装Performance Plugin

第二步：在需要集成jmeter-maven插件的工程配置页面，构建后操作加入一个Publish Performance test result report



第三步：在MavenBuild的Goals中填入verify



第四步：执行Jenkins构建，观察结果

### JMeter+Selenium

#### 使用JMeter JUnit Sampler

第一步：创建Selenium测试类

将selenium-server-standalone-2.37.0.jar（下载地址：<https://code.google.com/p/selenium/>）拷贝到%JMETER\_HOME%/lib/目录下，将Selenium测试用例打包成jar文件，并拷贝到%JMETER\_HOME%/lib/junit/目录下，注意：测试用例应该继承TestCase or SeleniumTestCase类，从而允许JMeter获取到该测试用例，且测试方法名称应该以test开头，示例如下：

|  |
| --- |
| Import java.util.concurrent.TimeUnit;    Import junit.framework.TestCase;    Import org.junit.After;  Import org.junit.Before;  Import org.junit.Test;  Import org.openqa.selenium.Alert;  Import org.openqa.selenium.By;  Import org.openqa.selenium.NoAlertPresentException;  Import org.openqa.selenium.NoSuchElementException;  Import org.openqa.selenium.WebDriver;  Import org.openqa.selenium.firefox.FirefoxDriver;  Import org.openqa.selenium.firefox.FirefoxProfile;    Public class Selenium extends TestCase {      Private WebDriver driver;      Private String baseUrl;      Private boolean acceptNextAlert = true;      Private StringBuffer verificationErrors = new StringBuffer();        @Before      Public void setUp() throws Exception {          FirefoxProfile profile = new FirefoxProfile();          driver = new FirefoxDriver(profile);          baseUrl = "[http://www.baidu.com"](http://www.baidu.com/);          driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);      }        @Test      Public void testSelenium() throws Exception {          driver.get(baseUrl + "/");          driver.findElement(By.id("kw")).clear();          driver.findElement(By.id("kw")).sendKeys("JMeter");          driver.findElement(By.id("su")).click();          driver.findElement(By.linkText("jmeter\_百度百科")).click();      }        @After      Public void tearDown() throws Exception {          driver.quit();          String verificationErrorString = verificationErrors.toString();          if(!"".equals(verificationErrorString)) {              fail(verificationErrorString);          }      }        Private boolean isElementPresent(By by) {          try{              driver.findElement(by);              returntrue;          } catch(NoSuchElementException e) {              returnfalse;          }      }        Private boolean isAlertPresent() {          try{              driver.switchTo().alert();              returntrue;          } catch(NoAlertPresentException e) {              returnfalse;          }      }        Private String closeAlertAndGetItsText() {          try{              Alert alert = driver.switchTo().alert();              String alertText = alert.getText();              if(acceptNextAlert) {                  alert.accept();              } else{                  alert.dismiss();              }              returnalertText;          } finally{              acceptNextAlert = true;          }      }  } |

第二步：JMeter加载Selenium测试类

在Jmeter中创建test group和JUnit sampler测试计划，并在JUnit sampler中选择测试用例的名称，选择测试方法并运行

#### 使用BeanShell Sampler

示例代码如下：

|  |
| --- |
| Import com.thoughtworks.selenium.\*;  Import org.junit.Before;  Import org.junit.Test;  Import org.junit.After;    Public class selenium extends SeleneseTestCase {      @Before      Public void setUp() throws Exception {          . . .      }      @Test      Public void testSelenium\_test() throws  Exception {          . . .      }      @After      Public void tearDown() throwsException {          selenium.stop();      }  } |

#### 使用JMeter Plugins的WebDriver Sampler

下载JMeterPlugins-WebDriver-1.1.2.zip（下载地址：<http://jmeter-plugins.org/downloads/all/>）并解压到JMETER\_HOME，然后添加jp@gc - Firefox Driver Config and jp@gc - Web Driver Sampler

## Loadrunner

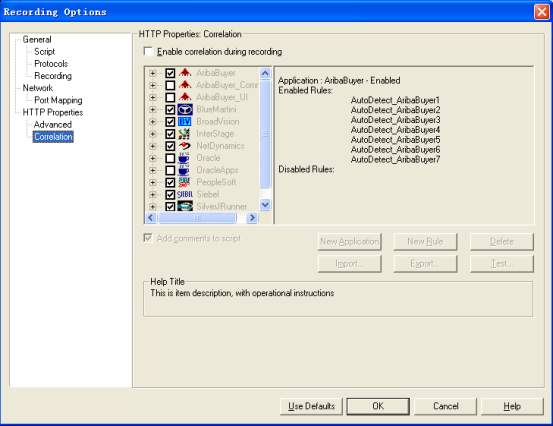
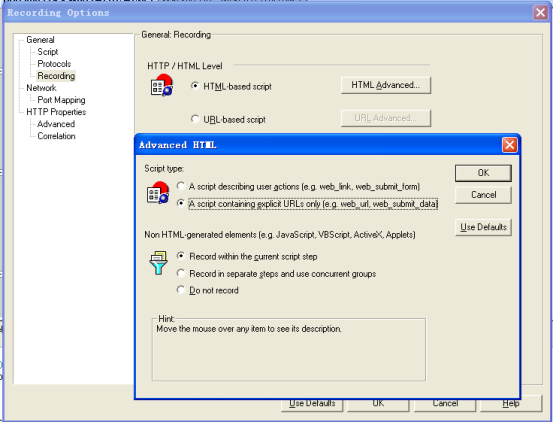
### loadrunner简介

LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试。通过使用LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。企业的网络应用环境都必须支持大量用户，网络体系架构中含各类应用环境且由不同供应商提供软件和硬件产品。难以预知的用户负载和愈来愈复杂的应用环境使公司时时担心会发生用户响应速度过慢，系统崩溃等问题。这些都不可避免地导致公司收益的损失。Mercury Interactive 的 LoadRunner 能让企业保护自己的收入来源，无需购置额外硬件而最大限度地利用现有的IT 资源，并确保终端用户在应用系统的各个环节中对其测试应用的质量，可靠性和可扩展性都有良好的评价。LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来帮助您更快的查找和发现问题。此外，LoadRunner 能支持广范的协议和技术，为您的特殊环境提供特殊的解决方案。

### 录制及调试脚本

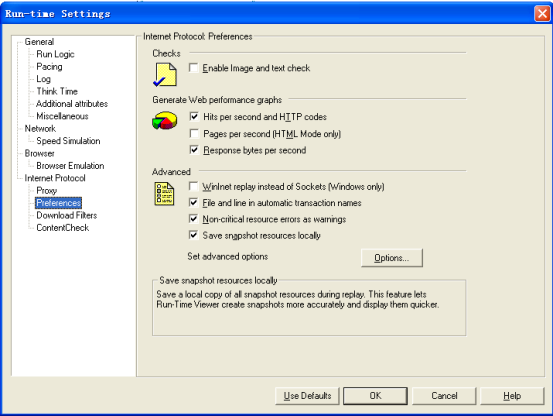
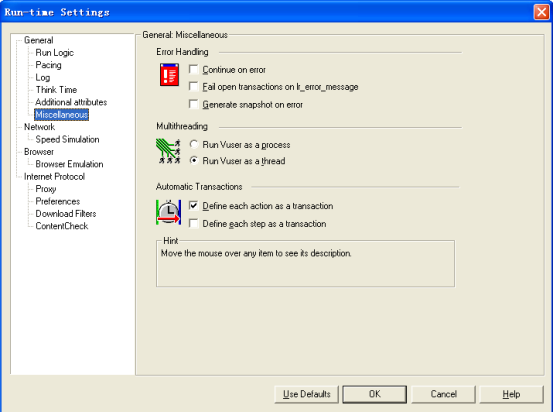
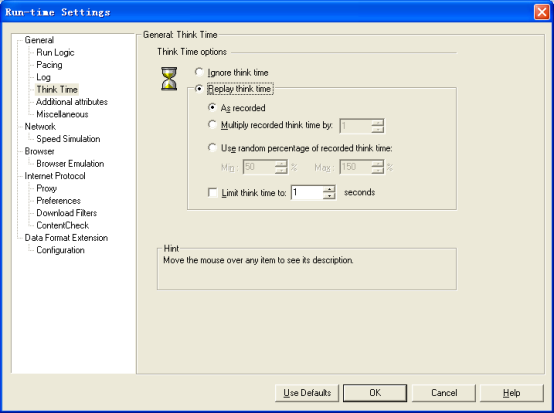
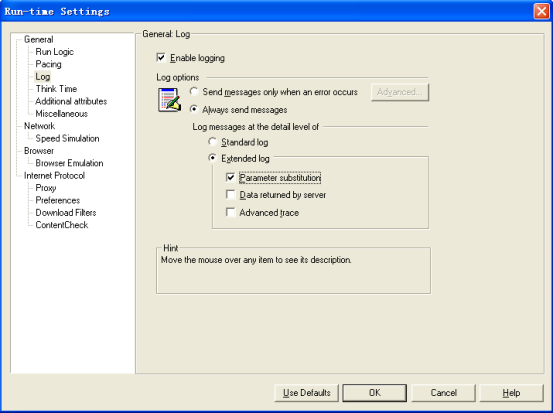
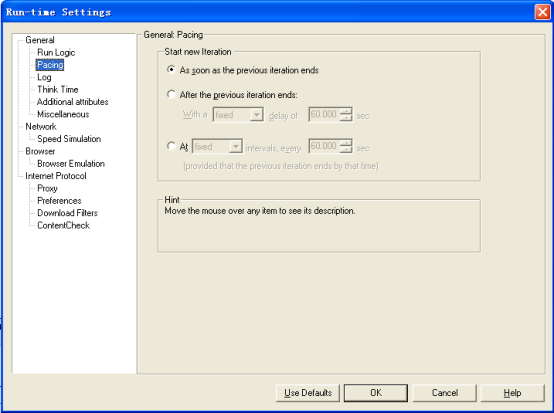
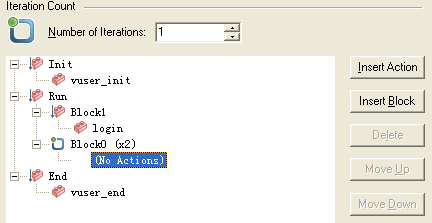
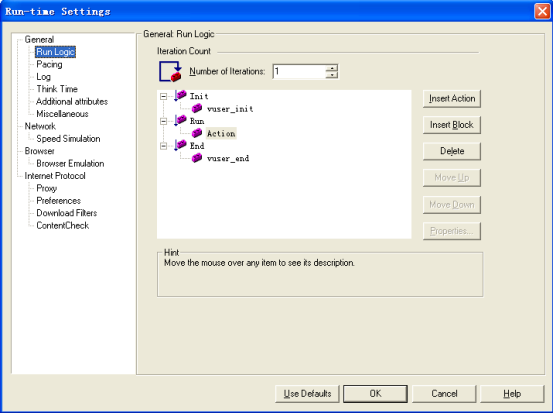
#### 录制脚本前的设置：

一般默认选择下面的。  
web URL 函数根据精确连接地址来找的。它不依赖于上下文。  
web link 类似QTP 先找文字再根据文字找连接。当所点击的字不在的时候，它就找不到了。它依赖于上下文  
  
  
关联，一般不做自动关联，即不选中Enable correlation



#### Run-time Setting设置

设置迭代：  
  
说明：  
Insert Action：通过插入action，可以把不同的业务放在不同的action中。只是设置迭代次数的时候，所有的action的迭代次数是相同的。  
Insert Block：如果需要把不同的业务设置成不同的迭代次数，只需要加入块,然后在块中设置迭代次数。  
  
  
 ：前一次迭代一完成就进行这次迭代。  
 ：前一次迭代60秒后，进行下一次迭代  
 ：每60秒迭代一次。前一次循环不一定完成，只要达到指定时间，就进入下一次循环中去。  
设置Pacing的目的：LoadRunner 模拟客户端向服务器发出请求，必须等待服务器对这个请求做出响应，并且客户端收到这个响应之后，才会重新发出新的请求，而服务器对请求的处理是需要一个时间的。测试场景以迭代的方式进行，如果不设置步进值的话，那么对于每个虚拟用户来说，每一个发到服务器的请求得到响应之后，会马上发送下一次请求。如果此时的服务器端的排队队列已满，服务器资源正处于忙碌的状态，那么该请求会驻留在服务器的线程中，换句话说，这个新产生的请求并不会对服务器端产生真正的负载，但该请求的计时器已经启动了，因此我们很容易就可以预见到，这个请求的响应时间会变得很长，甚至可能长到使得该请求由于超时而失败。等到测试结束后，我们查看一下结果，就会发现：事务平均响应时间很长，最小响应时间与最大响应时间的差距很大，而这个时候的平均响应时间，其实也就失去了它应有的意义。也就是说，由于客户端发送的请求太快而导致影响了实际的测量结果。  
因此，为了解决这个问题，我们可以在每两个请求之间插入一个间隔时间，这将会降低单个用户启动请求的速度。间歇会减少请求在线程中驻留的时间，从而提供更符合现实的响应时间。   
  
如果做参数化的时候，可以选择扩展日志，有利于调试脚本,但是在场景中要尽量关闭扩展日志，比较耗性能。（有些用到关联的时候，找左右边界，可以使用扩展日志）。  
  
Ignore think times：忽略思考时间，回放脚本时忽略所有的lr\_think\_times函数。  
Replay think time:  
按录制参数：回放期间，使用lr\_think\_times函数中显示的参数。来等待多长时间，单位秒。  
录制思考时间乘以：回放期间，使用lr\_think\_times函数中思考时间的倍数。  
使用录制思考时间的随机百分比：通过指定思考时间的范围，可以设置思考时间值的范围。  
将思考时间限制为：限制思考时间的最大值。如果此处设置的值与脚本中lr\_think\_times的值冲突时，以此处设置的时间为准。  
  
如果未选中Define each action as a transaction 在场景中就不计算init和action中的事务的时间，只计算自己加进去事务的时间。  
  
Options设置超时。报"Error -27728: Step download timeout (120 seconds)"错的时候在此处设置。如：HTTP-request connection timeout(sec)、 HTTP-request receive timeout(sec)、Step download timeout(sec)。把这些值设置大些。



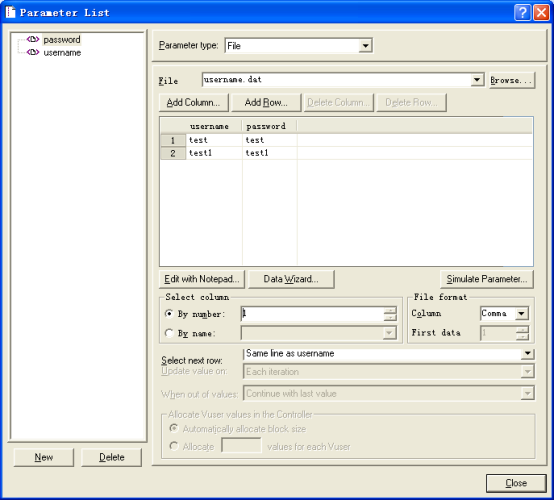
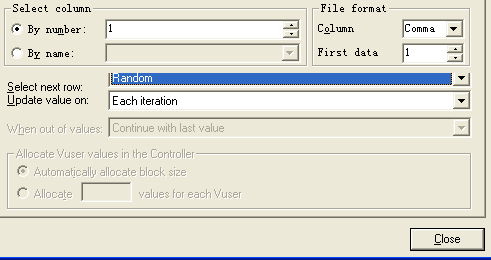
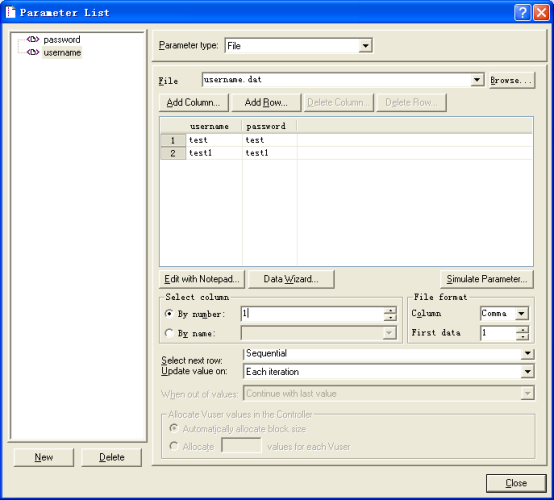
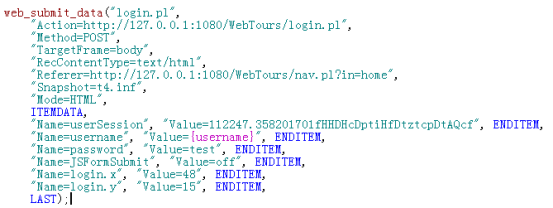
#### 其它设置

Tools->  ，可以设置场景  
Tools->   
Parameterzation 设置参数化时，左右符号，可以在这里面定义，默认为{}  
Display 选中Show browser during replay,回放的时候，显示浏览器  
Tools==  ，脚本改乱的时候，可以重新生成脚本。



#### 参数化

参数化就是在脚本中用参数取代常量值。下面以登录为例。  
1、打开脚本，找到登陆的脚本代码，如下图：  
  
2、参数化用户名，选中test，右键单击并选择"Replace with a parameter(替换为新参数)"  
3 、在弹出的对话框中，输入参数名username，并点击"确定"  
4 、参数化后，如下图：  
  
5 、参数化密码，与参数化用户名过程相同，或者点击工具栏上的Param List按钮打开参数设置页面。如下图：  
  
6、把用户名和密码放在同一个.dat文件中。  
可以通过点击"Add Row"按钮增加行，输入其他可以的值，点击"Add Column（添加列）"添加其他参数。  
  
7 、我们把login这个参数作为基准，设置它的select column（选择列）保持默认（1）；Select next row（选择下一行）选择：Random；Update value on（更新值得时间）选择：Each iteration。如下图：  
  
8、设置password参数，选择username.dat，使密码和用户名使用同一个文件。select column（选择列），在这里password选择2，Select next row（选择下一行）都选择：Same line as username，意思是和username取相同的行的数据，这样就可以保证一一对应了。下面是name参数的设置，如下图：



#### 检查点

检查点的作用是在回放脚本期间，搜索特定的文本字符串或者图片等内容，从而验证服务器响应内容的正确性。  
插入检查点的必要性：有的时候登陆操作，测试不报错，但是查看后台就报错。是没有插入检查点导致的！LoadRunner以用户名为角色向服务器发送一个登陆请求，却不会判断请求的返回消息是什么，只要有返回，即使这是个拒绝登陆的返回，LoadRunner也认为这个用户名登陆成功了，其实这个用户名登录是失败的。所以在登录或者[其他](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1969298)有重要页面跳转的地方，很有必要做检查点。  
函数：web\_find()和web\_reg\_find()。

#### 集合点

集合点，测试瞬间承受多大的压力，同步虚拟用户在同一时刻执行任务。集合点放在事物之前，如果放在事物之后，可能有些虚拟用户没达到集合点就开始计时，造成事物的时间统计不正确。  
函数：lr\_rendezvous("集合点名")。

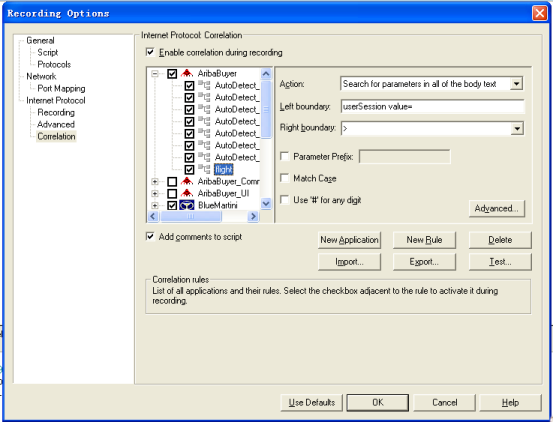
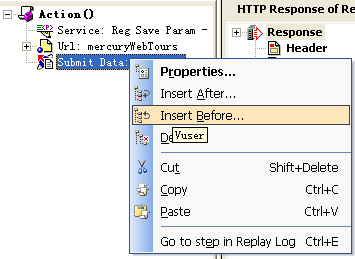
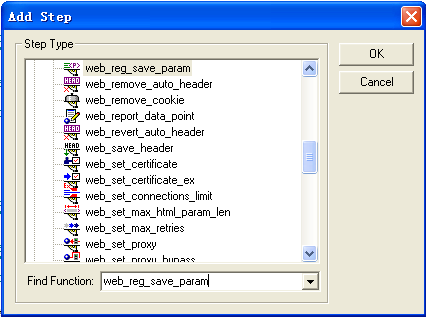
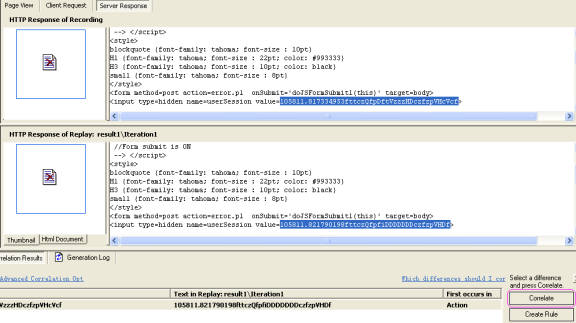
#### 事务

事物是要度量其服务器响应时间的任务或操作集。事物由开始、结束或多个业务操作/任务构成。事物必须成对出现，并且事物名称要相同。事物中不要将思考时间加进去，否则在回放思考时间设置被允许的情况下，思考时间将被算入事物的执行时间，从而影响了对事物正确执行时间分析与统计。  
如图：



#### 关联

在脚本回放过程中，客户端发出请求，通过关联函数所定义的左右边界值（也就是关联规则），在服务器所响应的内容中查找，得到相应的值，以变量的形式替换录制时的静态值，从而向服务器发出正确的请求，这种动态获得服务器响应内容的方法被称作关联。客户端的某个请求是随着服务器端的相应而动态变化的时候，我们就需要用到关联。  
例如：登录操作、先查后修改，先查后删除。关联有3种方式：自动关联、手动关联、设置关联规则  
**自动关联**Vuser==Scan script for correlations  
1、回放该脚本至少执行一次。  
2、点Scan script for correlations关联之后，脚本中自动生成web\_reg\_save\_param方法（切换到脚本视图）  
  
**手动关联**  
录制两个相同的脚本，通过Tools==compare with Vuser  
找出2次脚本发生变化的地方，找出它的左右边界，通过关联函数，进行关联，最后再把变化的值进行参数化。  
手动关联（一）  
通过上面的操作，在脚本视图下插入函数  
Insert==new stept   
  
手动关联—（二）  
在tree视图下插入函数  
  
手动关联—（三）  
在脚本中手动写关联函数  
web\_reg\_save\_param("sid",  
"LB=userSession value=","RB=>",LAST);  
web\_reg\_save\_param 为注册函数，应该放在请求之前  
**自定义关联规则**  
录制之前定好关联规则：  
Recording Options

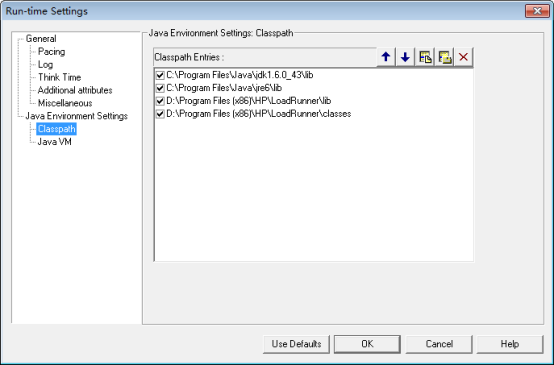
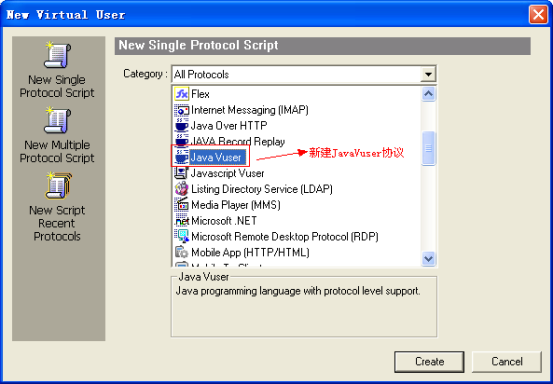


#### 思考时间

用户在执行两个连续操作期间等待的时间称为思考时间。Loadrunner在录制脚本时，VuGen将录制实际的停留等待时间，并将相应的等待时间插入到脚本。  
函数：lr\_think\_time(2）;  
思考时间可以通过运行时设置来决定是否启用思考时间，具体策略参见Run-time Setting中的thinktime

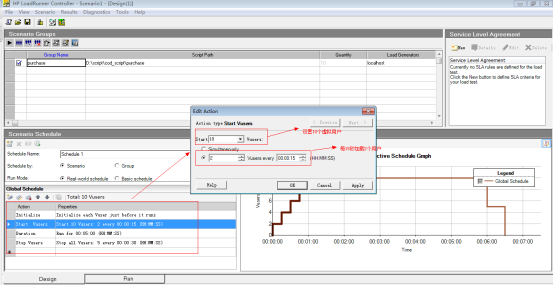
#### JavaVuser协议例子

1、新建JavaVuser协议  
  
2、调试脚本  
  
3、配置环境变量  
Run-time Settings->Java Environment Settings中的Classpath，如图：  
  
4、把需要引用的包放到脚本的根目录中。



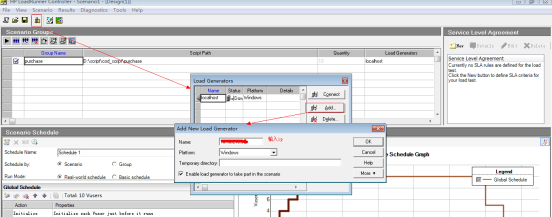
### 场景设置

#### 如何设置并发用户数



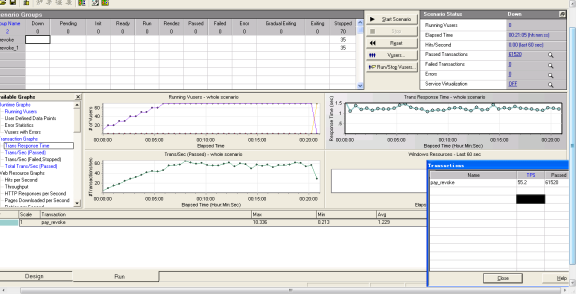
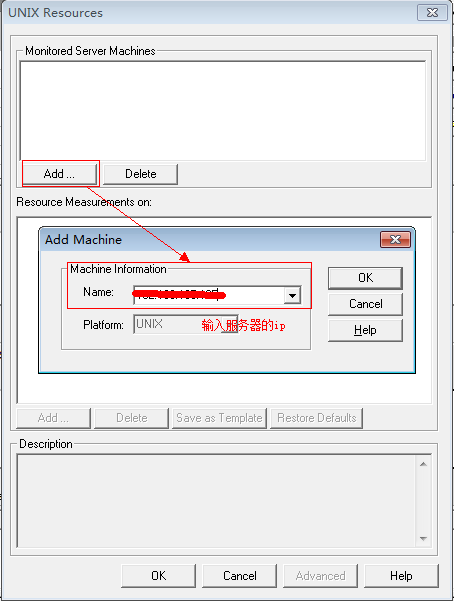
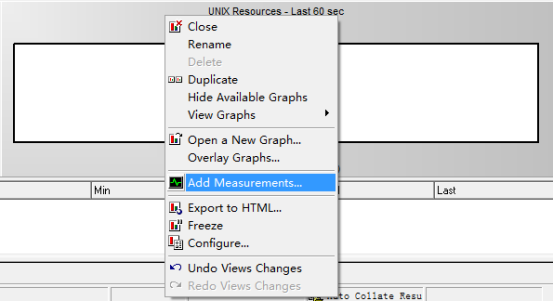
#### 设置压力机

注意:压力机要开启代理。



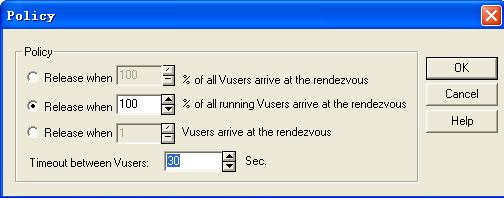
#### 监控系统资源情况

Run栏中  
  
  
点击  开始运行场景  
场景运行中



#### 集合点策略

1、所有用户到达集合点的时候释放  
2、百分之多少的正在运行的用户达到集合点时候释放  
3、多少个用户到达集合点的时候释放



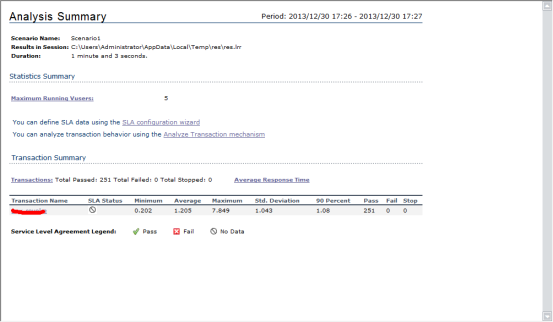
### 结果分析

场景运行完之后，点击  产生报告



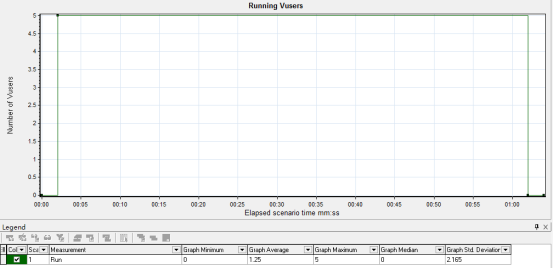
#### Analysis Summary

描述场景的基本信息、状态摘要信息、事物摘要信息等。通过Average和Std.deviation可以看出场景执行过程的稳定性。90Percent为90%的事物响应时间有多长。



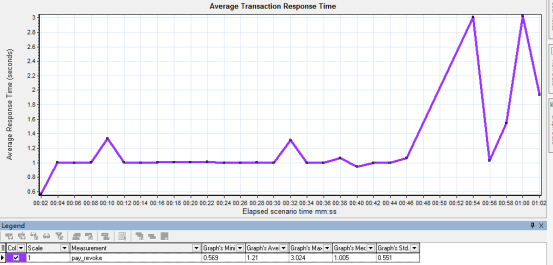
#### Running Vuser

说明虚拟用户加载规则，如Ramp up、Ramp down，持续多长时间。它描述场景的负载、即Vusers数量。  
Vusers数量会对吞吐量，点击率及被测服务器造成直接压力。当Vusers数量或者TPS没有达到需求时，需要增加Vusers数量。



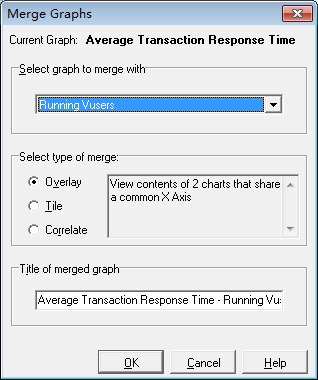
#### Transaction Response Time(Percentile)

显示事物的响应时间。



#### 合并图

在图标中右键Merge Graphs  
  
通过该方法可以对各个图标进行任意合并。



### 常见函数

lr\_get\_master\_host\_name 返回运行 LoadRunner Controller 的计算机名  
lr\_eval\_string 用参数的当前值替换参数  
lr\_save\_string 将以 NULL 结尾的字符串保存到参数中  
lr\_save\_var 将变长字符串保存到参数中  
lr\_save\_datetime 将当前日期和时间保存到参数中  
lr \_advance\_param 前进到下一个可用参数  
lr\_debug\_message 将调试信息发送到输出窗口  
lr\_error\_message 将错误消息发送到输出窗口  
lr\_get\_debug\_message 检索当前消息类  
lr\_log\_message 将消息发送到日志文件  
lr\_output\_message 将消息发送到输出窗口  
lr\_set\_debug\_message 设置调试消息类  
lr\_message 将消息发送到 Vuser 日志和输出窗口 lr\_load\_dll 加载外部 DLL  
lr\_continue\_on\_error 指定处理错误的方法 lr\_continue\_on\_error (0）  
web\_custom\_request 允许您使用 HTTP 支持的任何方法来创建自定义 HTTP 请求  
web\_image 在定义的图像上模拟鼠标单击 web\_link 在定义的文本链接上模拟鼠标单击   
web\_submit\_data 执行"无条件"或"无上下文"的表单  
web\_submit\_form 模拟表单的提交  
web\_url 加载由"URL"属性指定的 URL  
web\_find 在 HTML 页内搜索指定的文本字符串  
web\_image\_check 验证指定的图像是否存在于 HTML页内  
web\_reg\_find 在后面的 HTTP 请求中注册对 HTML源或原始缓冲区[中文](http://action.vogate.com/c/c.php?r=http%3A//www.baidu.com/s%3Fbs%3D%25C8%25ED%25BC%25FE+%25D0%25D4%25C4%25DC%25B2%25E2%25CA%25D4%26f%3D8%26wd%3Dloadrunner+%25BA%25AF%25CA%25FD&aid=5841&sid=6235007045049672&click=1&url=http%3A//www.seawww.com/style/info/domain.asp&v=0&k=%u4E2D%u6587&s=http%3A//old.blog.edu.cn/user2/35840/archives/2008/2119208.shtml&rn=119586)本字符串的搜索   
web\_add\_cookie 添加新的 Cookie 或修改现有的 Cookie  
web\_cleanup\_cookies 删除当前由 Vuser 存储的所有 Cookie  
web\_remove\_cookie 删除指定的 Cookie

## ****JVM监控工具****

### ****Java自带的性能监控工具****

| 命令名称 | 适用范围 | 使用方法 | 注意事项 |
| --- | --- | --- | --- |
| jstatd | Windows  Linux | jstatd -J-Djava.security.policy=jstatd.all.policy  -J-Djava.rmi.server.hostname=192.168.197.138  my.policy文件需要自己建立，内如如下： grant codebase "file:$JAVA\_HOME/lib/tools.jar" {  permission java.security.AllPermission; }; 这是安全策略文件，因为jdk对jvm做了jaas的安全检测，  所以我们必须设置一些策略，使得jstatd被允许作网络操作 | 由于我们server ip 和 hostname 映射不好，建议都添加   -J-Djava.rmi.server.hostname=192.168.197.138 |
| jps | Windows  Linux | jps 192.168.197.138，本机不用带 ip  列出远程服务器192.168.197.138 机器所有的jvm实例，采用rmi协议，默认连接端口为1099 （前提是远程服务器提供jstatd服务） | jps 是对应 jstatd 服务的。  相当于一个客户端 |
| jconsole | Windows  Linux | 一个图形化界面，可以观察到java进程的gc，class，内存等信息, 本机 java 进程都能直接监控。  如果是其他机器的， java 进程启动的时候必须添加 jmx 相关的jvm 参数：  -Dcom.sun.management.jmxremote.port=10100 \ -Dcom.sun.management.jmxremote.password.file=XXX/jconsole\_pwd/jmxremote.password \ -Dcom.sun.management.jmxremote.ssl=false \ -Djava.rmi.server.hostname=192.168.197.138 "  或不需要密码连接：  -Dcom.sun.management.jmxremote.port=10100 \ -Dcom.sun.management.jmxremote.authenticate=false \ -Dcom.sun.management.jmxremote.ssl=false \ -Djava.rmi.server.hostname=192.168.197.138 " | chmod 600 jmxremote.password  必须只读，要不会报错。  jmxremote.password  密码文件的设置和生成：  在$JRE\_HOME/lib/management下，  默认是只有一个jmxremote.password.template文件，  把这个文件拷贝成jmxremote.password,  修改最后2行指定用户名密码，如：monitorRole  123456  (用jconsole连接的时候就用这对用户名和密码) |
| jstat | Windows  Linux | jstat -class 2083 1000 10 （每隔1秒监控一次，一共做10次） 2083 是进程号  jstat -gc 2083 2000 20（每隔2秒监控一次，共做20）  具体参数：  -class：统计class loader行为信息 -compile：统计编译行为信息 -gc：统计jdk gc时heap信息 -gccapacity：统计不同的generations（包括新生区，老年区，permanent区）相应的heap容量情况 -gccause：统计gc的情况，（同-gcutil）和引起gc的事件 -gcnew：统计gc时，新生代的情况 -gcnewcapacity：统计gc时，新生代heap容量 -gcold：统计gc时，老年区的情况 -gcoldcapacity：统计gc时，老年区heap容量 -gcpermcapacity：统计gc时，permanent区heap容量 -gcutil：统计gc时，heap情况 -print compilation |  |
| jinfo | Windows  Linux | 观察运行中的java程序的运行环境参数：参数包括Java System属性和JVM命令行参数 实例：jinfo 2083 其中2083就是java进程id号，可以用jps得到这个id号 |  |
| jstack | Windows  Linux | 可以观察到jvm中当前所有线程的运行情况和线程当前状态 例： jstack 2083 |  |
| jmap | Windows  Linux | 观察运行中的jvm物理内存的占用情况。 参数如下： -heap：打印jvm heap的情况 -histo：打印jvm heap的直方图。其输出信息包括类名，对象数量，对象占用大小。 -histo：live ：同上，但是只答应存活对象的情况 -permstat：打印permanent generation heap情况  命令使用： jmap -heap 2083 可以观察到New Generation（Eden Space，From Space，To Space）,  tenured generation,Perm Generation的内存使用情况  生成 dump 文件：  jmap -[dump:format=b,file=heap.bin]( ) 16113 |  |

#### **具体案例：**

利用**jstack**找出 Jmeter在 linux 卡死的具体原因

(1)利用 **top** 找出占用 cpu 最高的 java 进程id：**8536**

(2)利用下面命令占用CPU 最高的线程Id： **8542**

     top -Hp **8536**-d 1 -n 1

(3)打印出java 进程 **8536**的堆栈信息：

**jstack** 8536 > javadump.txt

(4)  把线程id **8542**转换为十六进制 **215e**

在javadump.txt 中查找 **215e**, 然后就发现下面的堆栈信息：

      "StandardJMeterEngine" prio=10 tid=0x00002b94ec4a3000 nid=0x**215e** runnable [0x00002b9536fe0000]java.lang.Thread.State: RUNNABLE

         at org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient.teardownTest(AbstractJavaSamplerClient.java:66)

         at com.wangyin.jmeter.HessianRequest.teardownTest(HessianRequest.[java:140](http://java:140/))

        at org.apache.jmeter.protocol.java.sampler.JavaSampler.testEnded(JavaSampler.[java:287](http://java:287/))

         - locked<0x0000000740139670> (a java.util.HashSet)

         at org.apache.jmeter.engine.StandardJMeterEngine.notifyTestListenersOfEnd(StandardJMeterEngine.[java:221](http://java:221/))

        at org.apache.jmeter.engine.StandardJMeterEngine.run(StandardJMeterEngine.[java:437](http://java:437/))

        at java.lang.Thread.run(Thread.[java:662](http://java:662/))

### [Eclipse Mat Usage](http://10.45.241.159:8080/confluence/display/WYWiKi/Eclipse+Mat+Usage)

#### ****生成 Java heap Dump 文件****

Java –Xmx1024m …. **-XX:+HeapDumpOnOutOfMemoryError** ……, 当OOM(OutOfMemoryError) 发生是，就会生成一份 java dump 文件

交互式地获取Heap Dump， Java –Xmx1024m …. **-XX:+HeapDumpOnCtrlBreak** ……， 使用快捷键 ctrl + Break 去获取当前的 内存转储信息

利用 jmap 命令去获取 java dimp 文件

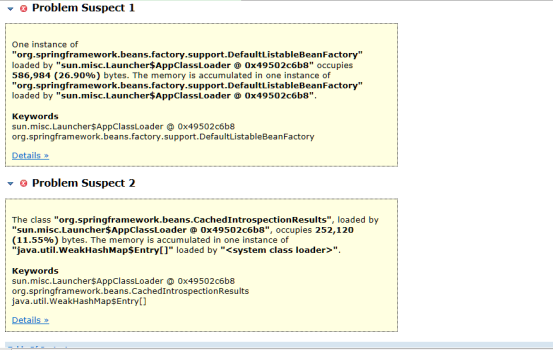
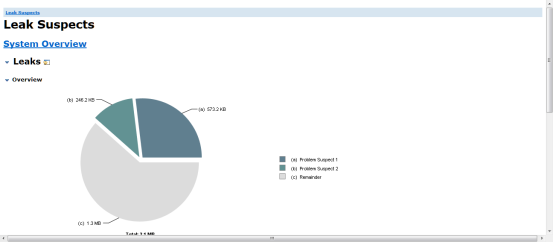
jmap -dump:format=b,file=heap.bin 4100 — 4100 是java 进程号

#### ****MAT安装****

下载地址: <http://www.eclipse.org/mat/downloads.php>   
安装方式: Update Site 或者 Stand-alone Eclipse RCP Applications  
使用手册: <http://www.eclipse.org/mat/1.3.0/noteworthy.html>   
**注意：** 由于目前 mat 最新版本是 1.30，对应 eclipse 版本是 3.7.2，而目前最新版本是 4.3。基于这个情况，如果本地eclipse 版本是大于 3.7.2，建议直接下载 mat 的**Stand-alone Eclipse RCP Applications** 版本 **[Windows (x86\_64) (46 MB)](http://www.eclipse.org/downloads/download.php?file=/mat/1.3.0/rcp/MemoryAnalyzer-1.3.0.20130517-win32.win32.x86_64.zip)** 。我尝试过在 eclipse 4.3 上通过 update site 方式安装，统计饼图无法安装成功。

在eclipse 安装完 mat 插件之后，直接打开生成的dump 文件，比如是 heap.bin

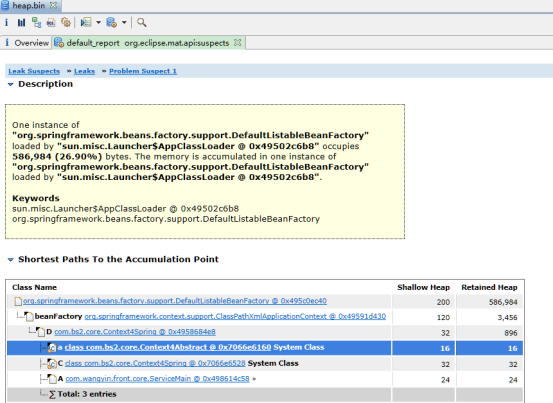
打开heap.bin



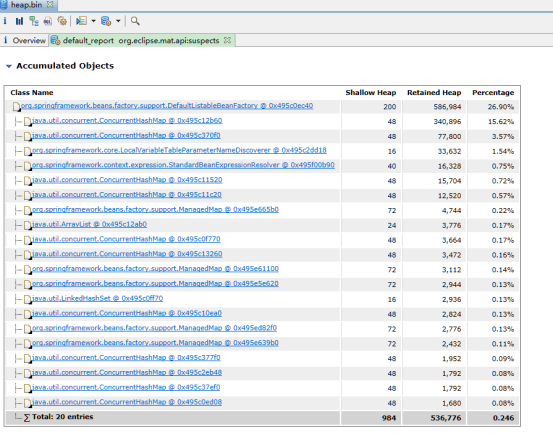
分析可能内存泄露的问题

点击 Problem Suspect 1 的 Details »，可以看到

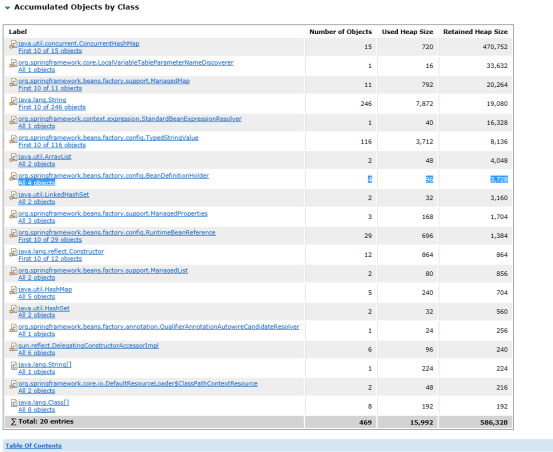
* 可能内存泄露问题的最短路径 Shortest Paths To the Accumulation Point



* 对象内存堆统计Accumulated Objects



* 每个类对象内存堆统计Accumulated Objects by Class



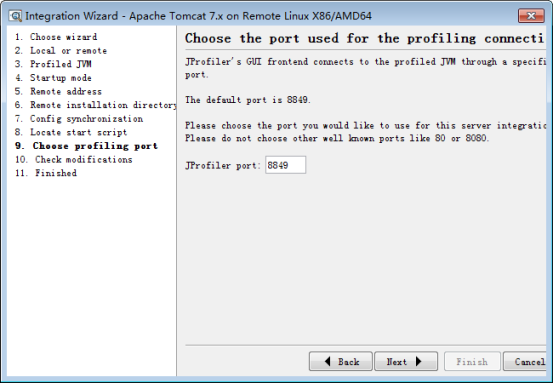
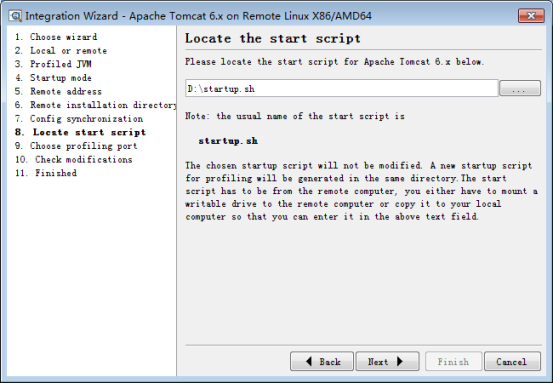
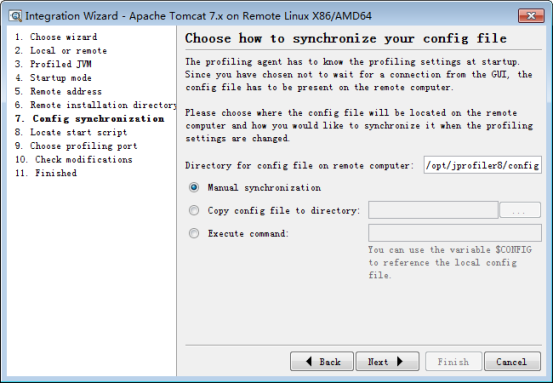
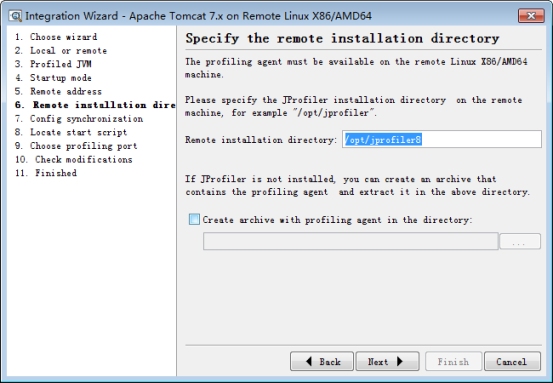
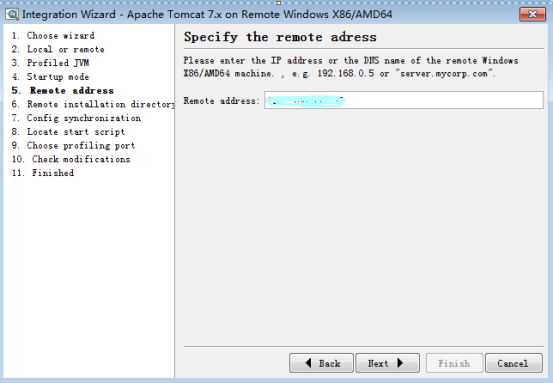
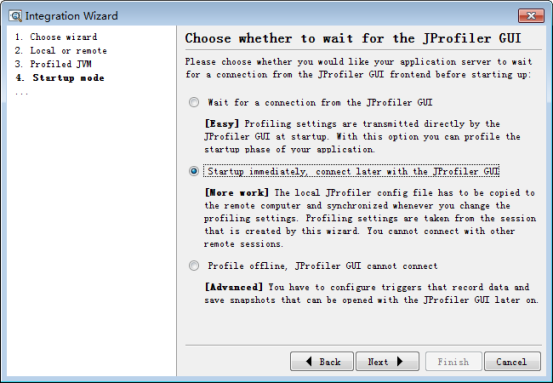
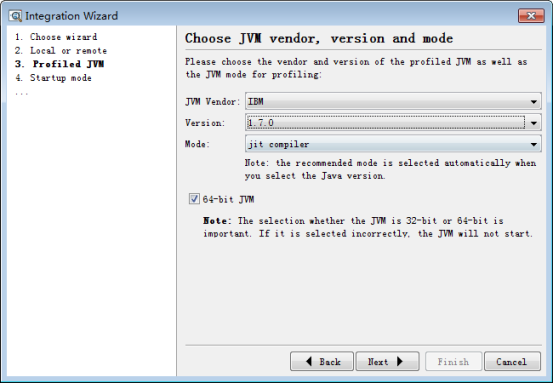
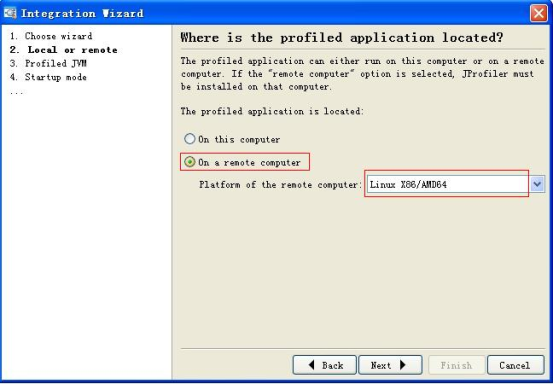
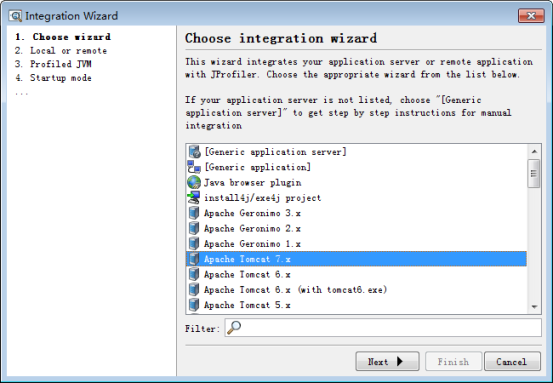
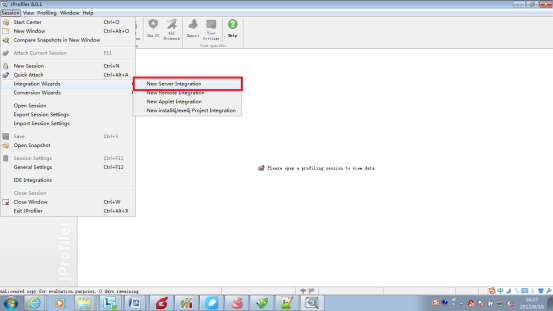
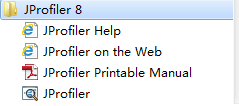
**参考资料** [memoryanalyzer Blog](http://dev.eclipse.org/blogs/memoryanalyzer/) [java类加载器体系结构](http://kenwublog.com/structure-of-java-class-loader) [ClassLoader](http://mindprod.com/jgloss/classloader.html)

### ****JProfiler****

#### ****JProfiler在linux上的配置使用****

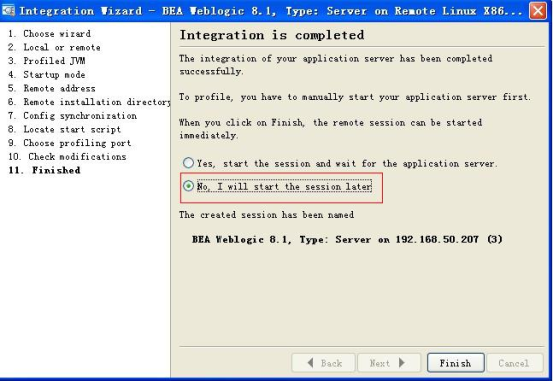
**注：该文章是我在使用JProfiler8.0.1版本时候的操作步骤**

**第一步：**下载JProfiler，下载地址为：[http://www.ej-technologies.com](http://www.ej-technologies.com/)  
我下载的是jprofiler\_windows-x64\_8\_0\_1这个版本，下载的时候要分别下载Windows版本和Linux版本。  
**第二步**：安装JProfiler-windows端。直接双击执行jprofiler\_windows-x64\_8\_0\_1.exe。（注：安装的过程要确保安装的机器上已经安装了JDK1.6或者JRE1.6）  
**第三步：**安装JProfiler-Linux端。将jprofiler\_linux\_8\_0\_1.sh拷贝到Linux的opt文件夹下，然后执行 chmod 777 jprofiler\_linux\_8\_0\_1.sh，确保jprofiler\_linux\_8\_0\_1.sh有可执行权限，然后通过  
./ jprofiler\_linux\_8\_0\_1.sh执行安装命令开始安装程序，程序会默认安装在/opt文件夹下，安装后一般为/opt/ jprofiler8。（安装的过程要确保安装的机器上已经安装了JDK1.6或者JRE1.6）  
**第四步：**启动windows端的JProfiler（启动前要将相应的防火墙及杀毒软件关闭），如图：  
  
启动后的界面如下，：  
  
**第五步：**开始配置相关的监控机器，在启动后的界面点Cancle按钮，然后点Session->Integration Wizards->New Server Integration来进行配置，具体操作如下图：  
  
**第六步：**开始正式的配置工作，在下图的界面选择要监视的软件，我这里选择的是:  
  
**第七步：**选择要监控的机器，看是监控本机的还是监控远程的，我这里是监控远程的一台Linux64位机器，如下图：  
  
**第八步：**选择java 虚拟机JVM的相关信息，该地方主要是中间件使用的JVM版本，我这里用的是IBM1.7版本。如果是用的JDK64位的，也就是JVM是64位的话，还需要勾选上**64 bit JVM.**如下图：  
  
**第九步：**选择启动的模式，如下图：  
  
**第十步：**配置远程要监控的中间件所在机器的地址，一般填写IP地址就可以，我这里是192..168.227.29，如下图：  
  
**第十一步：**配置JProfiler安装后的路径，我这里是/opt/jprofiler8，如下图：  
  
**第十二步：**选择config.xml文件将要存放在远程机器的位置，该文件一般需要放在JProfiler安装目录下的config文件夹下，我这里放在/opt/jprofiler8/config文件夹下。（config.xml来源是进行完整个JProfiler\_Windows端配置后在本机生成的config.xml文件，该文件的的位置在C:\Users\Administrator\.jprofiler8）如下图：  
/opt/jprofiler8/config  
  
**第十三步：**选择需要修改的tomcat应用服务的启动文件，该文件需要提前在远程部署好的应用服务拿下来存放到本地，如下图：  
D:\startup.sh    
**第十四步：**选择JProfiler的端口，该端口需要和服务端的保持一致，在这里我们一般不用修改，使用默认的端口就可以了，默认端口为8849。如下图：  
  
**第十五步：**系统会给出需要检查的配置信息：



1. 将本地C:\Users\Administrator\.jprofiler8\config.xml的config.xml文件拷贝到远程的/opt/jprofiler8/config下；
2. 本地会生成一个start\_jprofiler.sh文件，该文件和你拷贝的startup.sh在同一目录下，需要将startup\_jprofiler.sh拷到远程的应用服务下，路径也是和远程的startup.sh在同一文件夹下；

**第十六步：**配置完成，此处需要选择稍后启动配置好的Session,因为服务端我还没有进行相应的配置。如下图：  
  
**第十七步：**开始服务端的相关配置,也就是开始Linux端的配置，具体如下：  
17.1 将C:\Users\Administrator\.jprofiler8\config.xml文件拷贝到/opt/jprofiler8/config下；  
17.2 将生产的startup\_jprofiler.sh拷贝到应用服务下，我这里是D:\ startup\_jprofiler.sh ；  
17.3 修改startup.sh文件来设置JProfiler的HOME路径，  
-agentpath:/opt/jprofiler8/bin/linux-x64/libjprofilerti.so=port=8849,nowait,id=108,config=/opt/jprofiler8/config/config.xml  
注意：id要与config.xml中的id一致



## [OS与网络监控与分析工具](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1968350)

### ****工具总览****

top         查看进程活动状态以及一些系统状况

vmstat         查看系统状态、硬件和系统信息等

iostat         查看CPU 负载，硬盘状况

sar         综合工具，查看系统状况

mpstat         查看多处理器状况

netstat         查看网络状况

iptraf         实时网络状况监测

tcpdump         抓取网络数据包，详细分析

tcptrace         数据包分析工具

netperf         网络带宽工具

dstat         综合工具，综合了 vmstat, iostat, ifstat, netstat 等多个信息

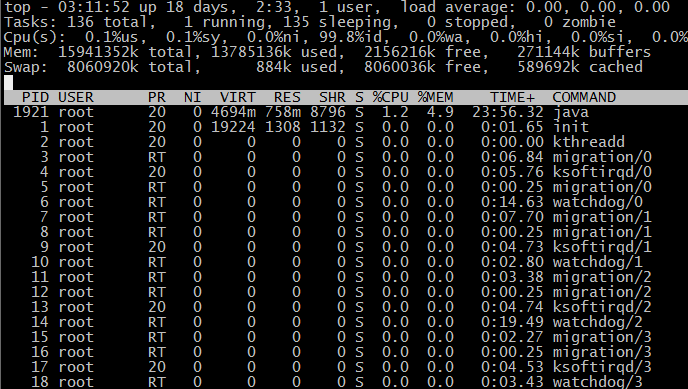
### top

1、类似于windows的任务管理器

2、f选择显示的列

3、P、M、T选择按cpu、mem、总时间排序

4、W输出到文件~/.toprc



### vmstat

r可运行队列的线程数，这些线程都是可运行状态，只不过 CPU 暂时不可用

b，被 blocked 的进程数，正在等待 IO 请求

swpd，已使用的 SWAP 空间大小，KB 为单位

free，可用的物理内存大小，KB 为单位

buff，物理内存用来缓存读写操作的 buffer 大小，KB 为单位

cache，物理内存用来缓存进程地址空间的 cache 大小，KB 为单位

si，数据从 SWAP 读取到 RAM（swap in）的大小，KB 为单位

so，数据从 RAM 写到 SWAP（swap out）的大小，KB 为单位

bi，磁盘块从文件系统或 SWAP 读取到 RAM（blocks in）的大小，block 为单位

bo，磁盘块从 RAM 写到文件系统或 SWAP（blocks out）的大小，block 为单位

in，被处理过的中断数

cs，系统上正在做上下文切换的数目

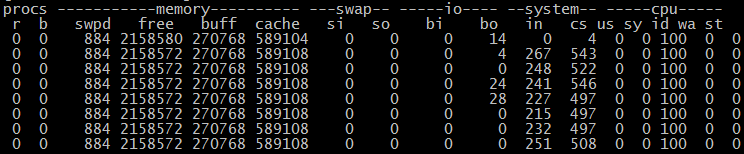
us，用户占用 CPU 的百分比

sy，内核和中断占用 CPU 的百分比

wa，所有可运行的线程被 blocked 以后都在等待 IO，这时候 CPU 空闲的百分比

id，CPU 完全空闲的百分比

#### 常用命令：vmstat 1



### iostat

rrqm/s: 每秒进行 merge 的读操作数目，即 delta(rmerge)/s

wrqm/s: 每秒进行 merge 的写操作数目，即 delta(wmerge)/s

r/s: 每秒完成的读 I/O 设备次数，即 delta(rio)/s

w/s: 每秒完成的写 I/O 设备次数，即 delta(wio)/s

rsec/s: 每秒读扇区数，即 delta(rsect)/s

wsec/s: 每秒写扇区数，即 delta(wsect)/s

rkB/s: 每秒读K字节数，是 rsect/s 的一半，因为每扇区大小为512字节。

wkB/s: 每秒写K字节数，是 wsect/s 的一半。

avgrq-sz: 平均每次设备I/O操作的数据大小 (扇区)，即 delta(rsect+wsect)/delta(rio+wio)

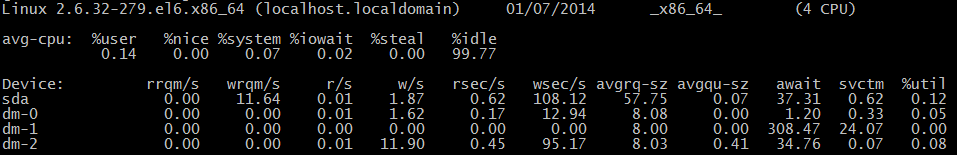
avgqu-sz: 平均I/O队列长度，即 delta(aveq)/s/1000 (因为aveq的单位为毫秒)。

await: 平均每次设备I/O操作的等待时间 (毫秒)，即 delta(ruse+wuse)/delta(rio+wio)

svctm: 平均每次设备I/O操作的服务时间 (毫秒)，即 delta(use)/delta(rio+wio)

%util: 一秒中有百分之多少的时间用于 I/O 操作，或者说一秒中有多少时间 I/O 队列是非空的，即 delta(use)/s/1000 (因为use的单位为毫秒)

#### 常用命令：iostat -x



### netstat

netstat –s

　　本选项能够按照各个协议分别显示其统计数据。如果我们的应用程序（如Web浏览器）运行速度比较慢，或者不能显示Web页之类的数据，那么我们就可以用本选项来查看一下所显示的信息。我们需要仔细查看统计数据的各行，找到出错的关键字，进而确定问题所在

netstat –e

　　本选项用于显示关于以太网的统计数据。它列出的项目包括传送的数据报的总字节数、错误数、删除数、数据报的数量和广播的数量。这些统计数据既有发送的数据报数量，也有接收的数据报数量。这个选项可以用来统计一些基本的网络流量）

netstat –r

　　本选项可以显示关于路由表的信息，类似于后面所讲使用route print命令时看到的信息。除了显示有效路由外，还显示当前有效的连接

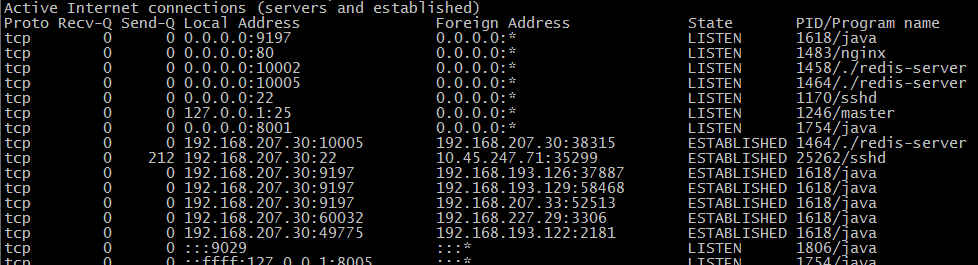
netstat –a

　　本选项显示一个所有的有效连接信息列表，包括已建立的连接（ESTABLISHED），也包括监听连接请求（LISTENING）的那些连接

netstat –n

　　显示所有已建立的有效连接

#### 常用命令：netstat -anltp



### dstat

1、默认收集-cpu-、-disk-、-net-、－paging-、-system-的数据，一秒钟收集一次

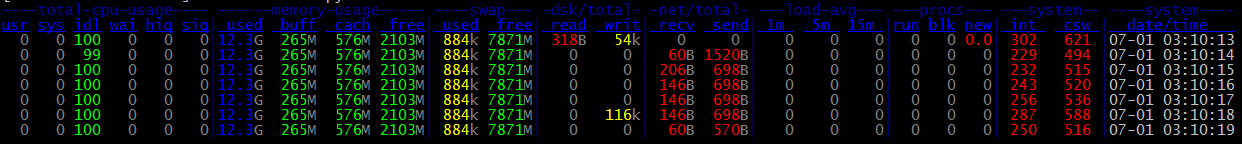
2、帮助dstat -h

3、常用dstat -cmsdnlpyt

4、显示当前占用cpu、memory、io最多的进程dstat --top-cpu --top-mem --top-io

5、导出文件dstat -cmsdnlpyt --output 1.csv

#### 常用命令：dstat -cmsdnlpyt



### Nmon

下载地址：<http://sourceforge.net/projects/nmon/files/>

安装：

tar zxvf nmon\_linux\_14i.tar.gz

cp nmon\_x86\_64\_centos6  nmon

cp nmon /bin/nmon

cp nmon /usr/bin/nmon

cp nmon /usr/local/bin/nmon

运行：nmon

帮助：nmon -h

例子：nmon -f -s 10 -c 100 -m /home/yangfei/Nmon/

说明：

1、它并不会消耗大量的 CPU 周期，通常低于百分之二，在更新的计算机上，其 CPU 使用率将低于百分之一

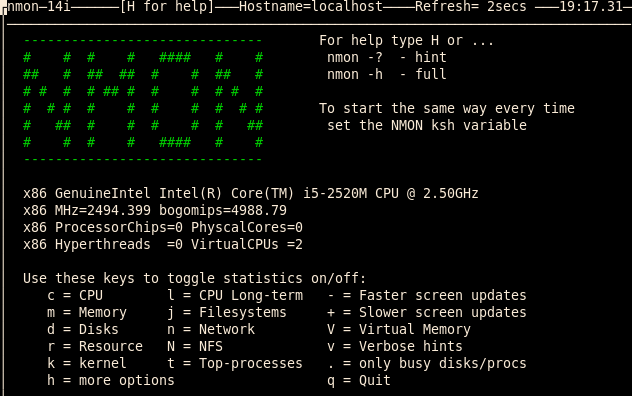
2、在屏幕上对数据进行显示，并且每隔两秒钟对其进行更新，然而，您可以很容易地将这个时间间隔更改为更长或更短的时间段

3、还可以将相同的数据捕获到一个文本文件，便于以后对报告进行分析和绘制图形，输出文件采用电子表格的格式 (.csv)

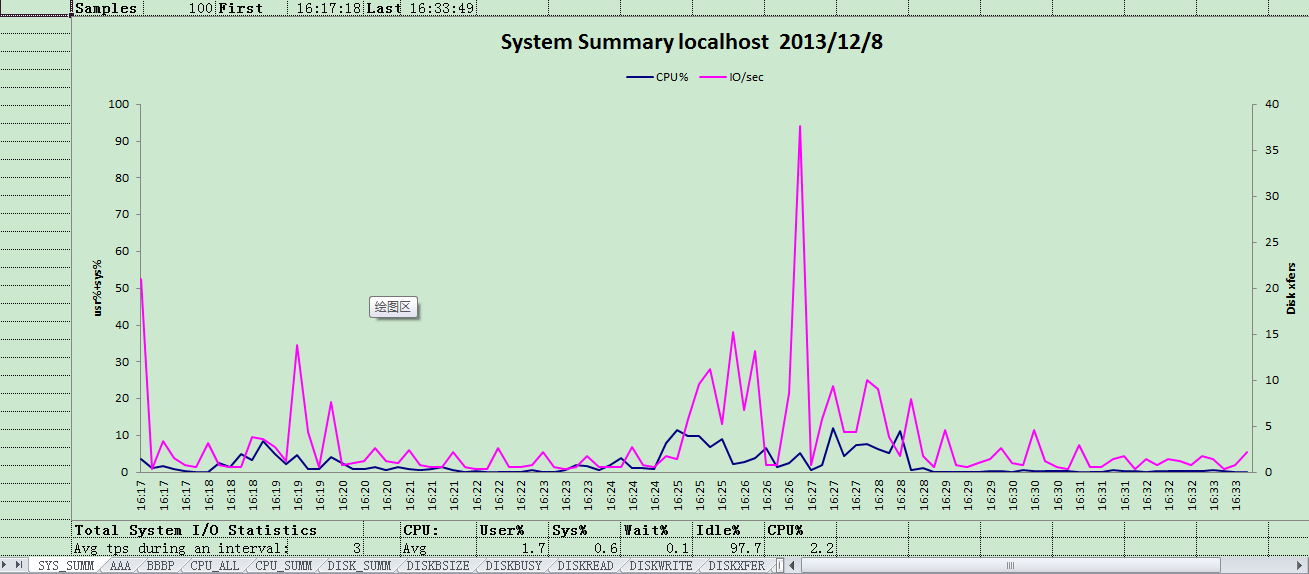
4、单次收集的文件大小最多不能超过65K行（EXCEL里的限制），大约是在nmon中 –c的值不超过330次

分析：nmon\_analyser打开.nmon文件生成.xlsx文件

nmon界面如下：



nmon analyser分析报告：



### 文件句柄数查看及修改

ulimit –a：查看所有限制值

ulimit –n：单个进程能够打开的最大文件句柄数量

lsof -n | awk '{print $2}' | sort | uniq -c | sort –nr：查看进程打开的句柄数，其中第一列是打开的文件句柄数量，第二列是进程号

lsof –n | grep process\_name：查看某进程打开了哪些句柄

lsof +d ./：查看当前目录下各文件被什么进程占用

ulimit -SHn 65535：即时修改限制数，重启失效

永久修改办法：

1、修改文件：/etc/security/limits.conf，添加：

\* soft nofile 32768 #限制单个进程最大文件句柄数（到达此限制时系统报警）

\* hard nofile 65536 #限制单个进程最大文件句柄数（到达此限制时系统报错）

2、修改文件：/etc/sysctl.conf，在文件中添加：

fs.file-max=655350 #限制整个系统最大文件句柄数

3、修改完成后，执行如下命令：

sbin/sysctl -p

使配置生效

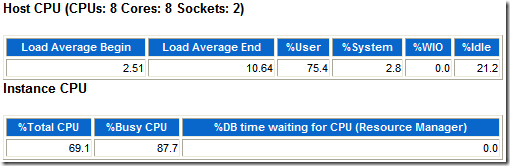
## [数据库分析与诊断工具](http://10.45.241.159:8080/confluence/pages/viewpage.action?pageId=1968356)

### ****Oracle AWR****

**ORACLE AWR报告生成和分析**  
Automatic Workload Repository是10g引入的一个重要组件。在里面存贮着近期一段时间内，默认是7天，数据库活动状态的详细信息。  
AWR报告是对AWR视图进行查询而得到的一份自动生成的报告。可以通过下面的脚本手工得到一份AWR报告。  
exec dbms\_workload\_repository.create\_snapshot;  
... running the specified workload  
exec dbms\_workload\_repository.create\_snapshot;  
@?/rdbms/admin/awrrpt  
通过AWR和AWR报告，DBA可以容易地获知最近数据库的活动状态，数据库的各种性能指标的变化趋势曲线，最近数据库可能存在的异常，分析数据库可能存在的性能瓶颈从而对数据库进行优化。  
AWR报告所有的数据来源于AWR视图，即以DBA\_HIST\_开头的所有系统表，Database Reference有对所有这些系统表的描述，这应该是Oracle官方对AWR报告的官方注释了。  
而对于如何有效地去分析AWR报告，这可能更需要DBA经验的日积月累。  
AWR的前身是Statspack，Statspack在10g和11g中也有提供，同时和AWR一起做了同步更新，而且Statspack是公开源代码的，因此，关于Statspack的资料，还有Statspack的源代码，都是理解AWR的一个有用的辅助。  
如果关注数据库的性能，那么当拿到一份AWR报告的时候，最想知道的第一件事情可能就是系统资源的利用情况了，而首当其冲的，就是CPU。  
而细分起来，CPU可能指的是

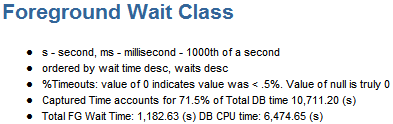
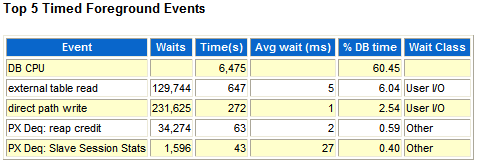
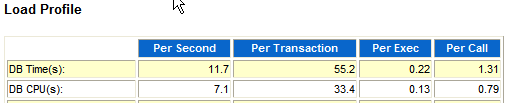
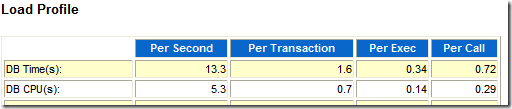
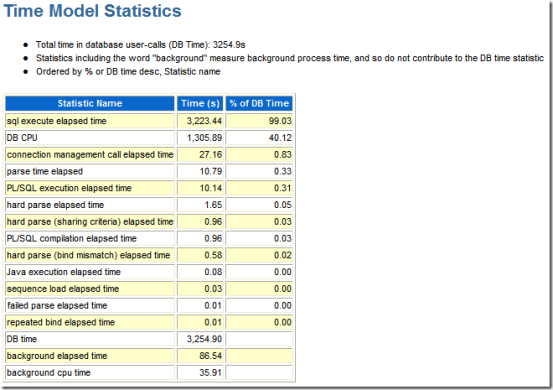
* OS级的User%, Sys%, Idle%
* DB所占OS CPU资源的Busy%
* DB CPU又可以分为前台所消耗的CPU和后台所消耗的CPU

如果数据库的版本是11g，那么很幸运的，这些信息在AWR报告中一目了然：  
  
OS级的%User为75.4，%Sys为2.8，%Idle为21.2，所以%Busy应该是78.8。DB占了OS CPU资源的69.1，%Busy CPU则可以通过上面的数据得到：  
%Busy CPU = %Total CPU/(%Busy) \* 100 = 69.1/78.8 \* 100 = 87.69，和报告的87.7相吻合。  
如果是10g呢，则需要手工对Report里的一些数据进行计算了。Host CPU的结果来源于DBA\_HIST\_OSSTAT，AWR 报告里已经帮忙整出了这段时间内的绝对数据(这里的时间单位是centi second，也就是1/100秒)。  
  
这里，  
%User=USER\_TIME/(BUSY\_TIME+IDLE\_TIME)\*100=146355/(152946+41230)\*100 = 75.37  
%Sys = SYS\_TIME/(BUSY\_TIME+IDLE\_TIME)\*100  
%Idle = IDLE\_TIME/(BUSY\_TIME+IDLE\_TIME)\*100  
值得注意的，这里已经隐含着这个AWR报告所捕捉的两个snapshot之间的时间长短了。有下面的公式  
BUSY\_TIME + IDLE\_TIME = ELAPSED\_TIME \* CPU\_COUNT  
正确的理解这个公式可以对系统CPU资源的使用及其度量的方式有更深一步的理解。因此ELAPSED\_TIME = (152946+41230)/8/100 = 242.72 seconds。  
至于DB对CPU的利用情况，这就涉及到10g新引入的一个关于时间统计的视图了， v$sys\_time\_model，简单而言，Oracle采用了一个统一的时间模型对一些重要的时间指标进行了记录，具体而言，这些指标包括：



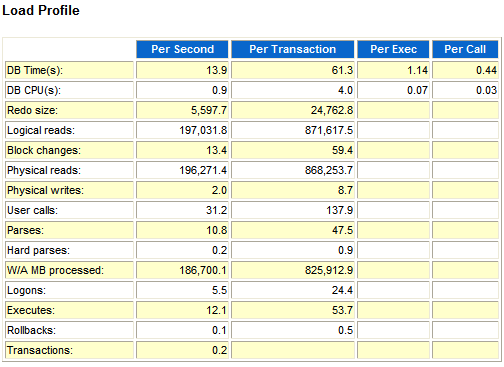
* background elapsed time
  + background cpu time
    - RMAN cpu time (backup/restore)
* DB time
  + DB CPU
  + connection management call elapsed time
  + sequence load elapsed time
  + sql execute elapsed time
  + parse time elapsed
    - hard parse elapsed time
      * hard parse (sharing criteria) elapsed time
      * hard parse (bind mismatch) elapsed time
    - failed parse elapsed time
      * failed parse (out of shared memory) elapsed time
  + PL/SQL execution elapsed time
  + inbound PL/SQL rpc elapsed time
  + PL/SQL compilation elapsed time
  + Java execution elapsed time
  + repeated bind elapsed time

我们这里关注的只有和CPU相关的两个: background cpu time 和 DB CPU。这两个值在AWR里面也有记录:  
  
Total DB CPU = DB CPU + background cpu time = 1305.89 + 35.91 = 1341.8 seconds，再除以总的 BUSY\_TIME + IDLE\_TIME：  
% Total CPU = 1341.8/1941.76 = 69.1%，这刚好与上面Report的值相吻合。  
其实，在Load Profile部分，我们也可以看出DB对系统CPU的资源利用情况。  
  
用DB CPU per Second除以CPU Count就可以得到DB在前台所消耗的CPU%了。这里 5.3/8 = 66.25 %，比69.1%稍小，说明DB在后台也消耗了大约3%的CPU。  
DB CPU是一个用于衡量CPU的使用率的重要指标。假设系统有N个CPU，那么如果CPU全忙的话，一秒钟内的DB CPU就是N秒。  
如何去表征一个系统的繁忙程度呢？除了利用CPU进行计算外，数据库还会利用其它计算资源，如网络，硬盘，内存等等，这些对资源的利用同样可以利用时间进行度量。假设系统有M个session在运行，同一时刻，有的session可能在利用CPU，有的session可能在访问硬盘，那么，在一秒钟内，所有session的时间加起来就可以表征系统在这一秒内的繁忙程度，一般的，这个和的最大值应该为M。这其实就是Oracle提供的另一个重要指标：DB time，它用以衡量前端进程所消耗的总时间。  
对除CPU以后的计算资源的访问，Oracle用等待事件进行描述。同样地，和CPU可分为前台消耗CPU和后台消耗CPU一样，等待事件也可以分为前台等待事件和后台等待事件。  
DB Time一般的应该等于DB CPU + 前台等待事件所消耗时间的总和。等待时间通过v$system\_event视图进行统计，DB Time和DB CPU则是通过同一个视图，即v$sys\_time\_model进行统计。  
Load Profile一节就有了对DB Time的描述：  
  
这个系统的CPU个数是8，因此我们可以知道前台进程用了系统CPU的7.1/8=88.75%。DB Time/s为11.7，可以看出这个系统是CPU非常繁忙的。里面CPU占了7.1，则其它前台等待事件占了11.7 – 7.1 = 4.6 Wait Time/s。DB Time 占 DB CPU的比重呢？ 7.1/11.7= 60.68%  
Top 5 Timed Events，或许很多人都对它有所耳闻，按照CPU/等待事件占DB Time的比例大小，这里列出了Top 5。如果一个工作负载是CPU繁忙型的话，那么在这里应该可以看到 DB CPU的影子。  
  
注意到，我们刚刚已经算出了DB CPU 的%DB time，60%。其它的external table read, direct path write, PX Deq: read credit, PX Deq: Slave Session Stats这些就是占比重40的等待事件里的Top 4了。  
回过头再再研究下这个Top 5 Timed Foreground Events，如果先不看Load Profile，你能说出这个一个CPU-Bound的工作负载吗？  
答案是否定的，要知道系统CPU的繁忙程序，还要知道这个AWR所基于两个snapshot的时间间隔，还要知道系统CPU的个数。要不，系统可以是一个很IDLE的系统呢。记住CPU利用率 = DB CPU/(CPU\_COUNT\*Elapsed TIME)。  
这个Top 5 给我们的信息只是这个工作负载应该是并行查询，从外部表读取数据，并用insert append的方式写入磁盘，同时，主要时间耗费在CPU的运算上。上面提到，DB Time一般的应该等于DB CPU + 前台等待事件所消耗时间的总和。在下面有对这三个值的统计：



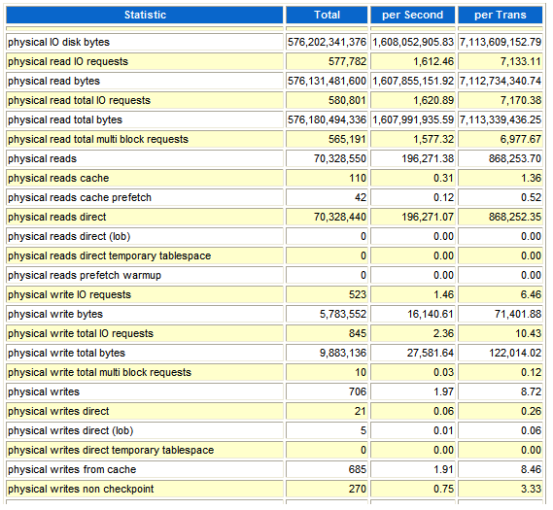
* DB CPU = 6474.65
* DB TIME = 10711.2
* FG Wait Time = 1182.63

明显的，DB CPU + FG Wait Time < DB Time，只占了71.5%，其它的28.5%被消耗到哪里去了呢？这里其实又隐含着一个Oracle如何计算DB CPU和DB Time的问题。当CPU很忙时，如果系统里存在着很多进程，就会发生进程排队等待CPU的现象。在这样，DB TIME是把进程排队等待CPU的时间算在内的，而DB CPU是不包括这一部分时间。这是造成 DB CPU + FG Wait Time < DB Time的一个重要原因。如果一个系统CPU不忙，这这两者应该就比较接近了。  
不要忘了在这个例子中，这是一个CPU非常繁忙的系统，而71.5%就是一个信号，它提示着这个系统可能是一个CPU-Bound的系统。  
除了DB CPU，DB Time，或许另一个比较常用的指标应该是IO的利用情况。关于IO的指标就比较多了，单单在Load Profile里面就有5个，在DB Time和DB CPU的下面：  
  
这5个指标的值都来自v$systat视图，分别是：



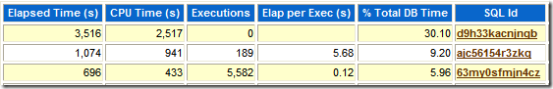
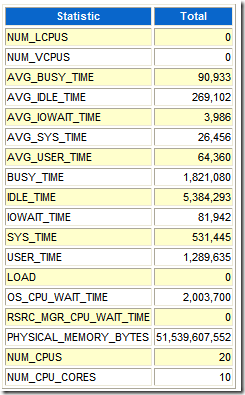
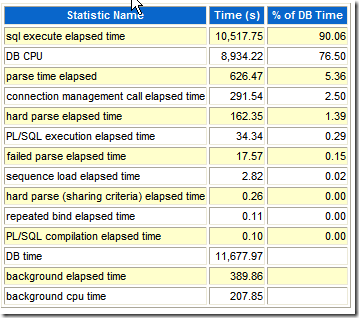
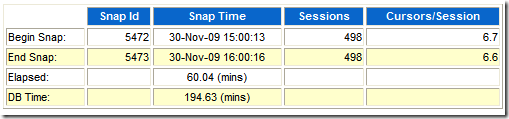
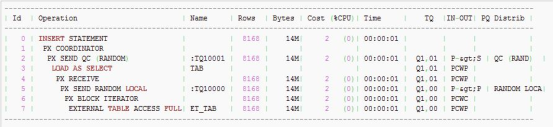
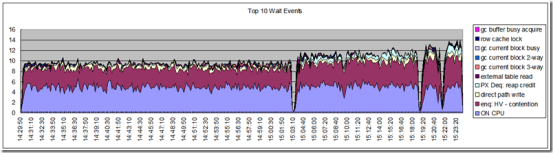
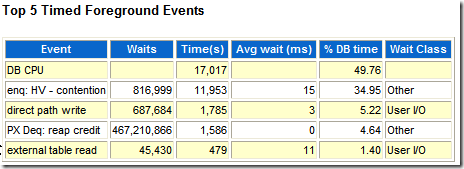
* Redo Size: 'redo size'
* Logical reads = 'session logical reads' or ('db block gets' + 'consistent gets')
* Blocks Changes = 'db block changes'
* Physical reads = 'physical reads'
* Physical writes = 'physical writes'

如何得到系统大致的MBPS呢？  
MBPS=(Physical reads+Physical writes)\*Block\_Size  
= (196,271.4+2.0)\*8\*1024/1024/1024 = 1533 MB/s  
更准确的MBPS可以从Instance Activity Stats部分获得。  
  
physical IO disk bytes = physical read total bytes + physical write total bytes  
值得注意的是这里physical write total bytes大致是physical write bytes的两倍。这应该是physical write total bytes统计的是磁盘的IO，而这里，我们做了ASM，normal redundancy，一份数据写了两遍的原因。  
Load Profile剩下的部分主要是关于各种执行情况的统计，除了W/A MB processed来自v$pgastat（单位其实也是Byte，不是MB），其它数据都是来自于v$sysstat。



* Blocks Changes: 'db block changes'
* User calls: 'user calls'
* Parses: 'parse count (total)'
* Hard parses: 'parse count (hard)'
* Logons: 'logons cumulative'
* Executes: 'execute count'
* Rollbacks: 'user rollbacks'
* Tranasactions: 'user rollbacks' + 'user commits'
* W/A MB processed: 'bytes processed'

一般而言，Hard parses < Parses < Executes < User Calls。  
AWR的一般性介绍我想差不多就这些了，其它部分的介绍借助于一些更具体的AWR报告进行分析可能会更加方便和清晰。  
如果这个系列是按"总-分-总"组织的话，接下来的系列应该是进行"分"这一部分了。  
构建DSS系统的第一步离不开数据加载，通过文本文件加载是最常见的方式，Oracle提供了外部表加载的方法，即把一个文本文件当成一个正常的表来进行操作，通过类似insert /\*+ append \*/ into table select from external\_table的方式进行加载。  
数据加载是一个CPU-Bound的过程，不过是通过什么工具，external table也好，sqlldr也好，imp也好，impdp也好。换句话说，如果连数据加载都出现I/O瓶颈，这个系统的配置就说不过去了。  
这个过程的AWR报告会是怎么样子的呢？  
先做个一般的假定，从外部表加载数据到一个本地分区表。  
Top 5 Timed Events类似下面：  
  
如果去抓取这段时间DBA\_HIST\_ACTIVE\_SESS\_HISTORY的数据，并转换为图表的话，我们会得到更形象的Top 10 Wait Events。  
  
enq: HV – contention是什么东西呢？  
在11.2以前，对于分区表的parallel direct-path load，Oracle采用的是brokered load的方式，即所有的PX Slaves共享对每个分区的high water mark的访问，通过轮流持有high water mark实现对每个segment添加新的blocks。这种方法对于充分利用extent的空间是有帮助的，不过带来的问题就是对high water mark的竞争，也就是这里的enq: HV – contention。在执行计划中，这以RANDOM LOCAL 标记。下面是一个例子：  
  
一个好消息是，11.2引入了一种新的方式，叫做PKEY distribution。在这种方式下，一个特定的分区只交给一个或多个特定的PX slave负责，这种方式不仅减少了对high water mark的争用，而且可以实现partition内更好的压缩率。  
现在探讨另一个对系统CPU进行度量的指标: CPU used by this session。  
他刚好有一份AWR报告，在这份报告里，出现了严重的CPU used by this session和DB CPU不一致的现象。  
下面是这份报告的一些片断：  
  
  
  
  
再做进一步的归纳：  
OS Busy% = 1821080/(1821080+5384293) = 25%  
Inst CPU% (using DB CPU) = 8934.22\*100/(1821080+5384293)=12%  
Inst CPU% (using CPU used by this session) = 418035/(1821080+5384293) = 6%  
用CPU used by this session计算出的CPU利用率竟然只是用DB CPU计算出来的利用率的一半！  
"prior to 10g Oracle usually updated time figures at the end of each database call; but from 10g there are some views where time is updated more regularly.   
The "DB CPU" from v$sess\_time\_model increases every six seconds, while the "CPU used by this session" from v$sesstat changes only at the end of the test."  
如何验证这一点呢？在浏览这份报告的TOP SQL时，我们发现了下面的现象：  
  
这是从SQL ordered by Elapsed Time截取出来的Top 3 SQL。TOP 1的SQL用了DB Time的30.10%，用了2517s 的CPU Time。但请注意它的Executions的值却为0。也就是说，这里的CPU Time是还没有被计算入CPU used by this session这个指标里面的。  
我们再把2517s加回来，看出误差缩小多少:(251700+418035)/(1821080+5384293) = 9%  
这时和用DB CPU计算出来的12%还是有1/4的差距。  
从这个例子可以看出，用DB CPU度量还是比用CPU used by this session来得准确的。特别在有大查询在跑的过程中抓的AWR，这个误差很有可能会被放大。

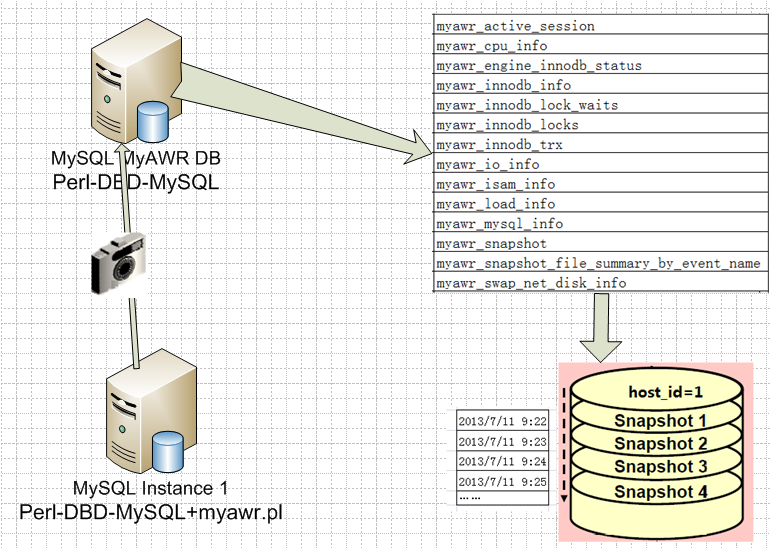
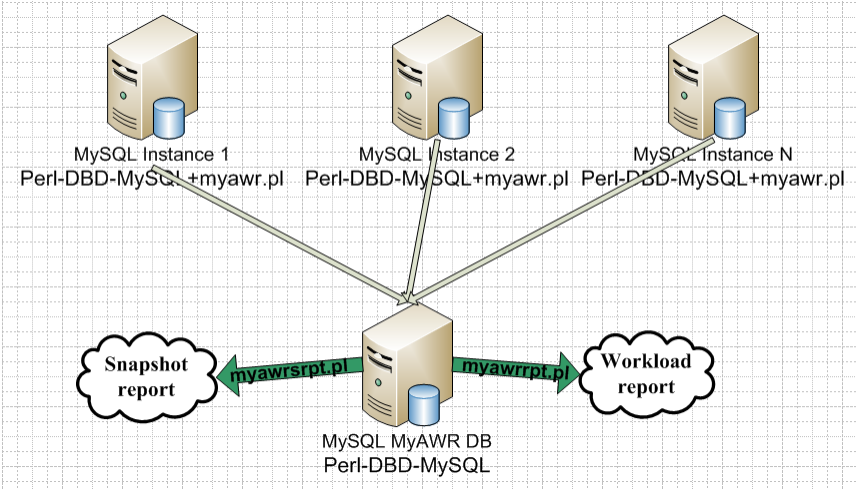


### ****MySQL AWR****

#### ****What is myawr****

开源地址见：[{+}](https://github.com/noodba/myawr)<https://github.com/noodba/myawr+>  
Myawr is a tool for collecting and analyzing performance data for MySQL database (including os info ,mysql status info and Slow Query Log all of details).   
The idea comes from Oracle awr. Myawr periodic collect data and save to the database as snapshots.  
Myawr was designed as CS architecture.Myawr depends on (but not necessary) performance schema of MySQL database.   
Myawr consists of three parts:  
myawr.pl--------a perl script for collecting mysql performance data  
myawrrpt.pl-----a perl script for analyzing mysql performance data  
myawrsrpt.pl-----a perl script for analyzing mysql peak time data   
Myawr relies on the Percona Toolkit to do the slow query log collection.  
Specifically you can run pt-query-digest. To parse your slow logs and insert them into your server database for reporting and analyzing.   
Thanks to orzdba.pl (zhuxu@taobao.com).

Here is myawr architecture: 



#### ****Myawr Data Model****

myawr db include tables list:  
mysql> show tables;  
----------------------------------------------------------  
| Tables\_in\_myawr |  
----------------------------------------------------------  
| myawr\_active\_session |  
| myawr\_cpu\_info |  
| myawr\_engine\_innodb\_status |  
| myawr\_host |  
| myawr\_innodb\_info |  
| myawr\_innodb\_lock\_waits |  
| myawr\_innodb\_locks |  
| myawr\_innodb\_trx |  
| myawr\_io\_info |  
| myawr\_isam\_info |  
| myawr\_load\_info |  
| myawr\_mysql\_info |  
| myawr\_query\_review |  
| myawr\_query\_review\_history |  
| myawr\_snapshot |  
| myawr\_snapshot\_events\_waits\_summary\_global\_by\_event\_name |  
| myawr\_snapshot\_file\_summary\_by\_event\_name |  
| myawr\_swap\_net\_disk\_info |  
----------------------------------------------------------  
18 rows in set (0.00 sec)   
some key tables:  
myawr\_host-- mysql instance config table  
myawr\_snapshot – snapshot table,exec myawr.pl a time as a shapshot  
myawr\_query\_review\_history – The table in which to store historical values for review trend analysis about slow log.   
myawr data model:  
myawr\_snapshot.host\_id reference myawr\_host.id;  
myawr\_query\_review\_history.hostid\_max reference myawr\_host.id;  
myawr\_innodb\_info.(host\_id,snap\_id) reference myawr\_snapshot.(host\_id,snap\_id);

#### ****Quickstart****

If you are interesting to use this tool, here's what you need:   
1. A MySQL database to store snapshot data and slow log analysis data .  
2. pt-query-digest by percona  
3. A MySQL server(version 5.5) with perl-DBD-mysql  
4. slow query logs named like slow\_20130521.log,you can switch slow logs every day.   
3.1 install db(where you store shapshot data,perl-DBD-MySQL is required)  
Connect to the MySQL database where store the performance data and issue the following command in myawr.sql:  
grant all on myawr.\* to 'myuser'@'localhost' identified by "111111";  
grant all on myawr.\* to 'myuser'@'%' identified by "111111";  
then create tables.   
3.2 initialize myawr\_host(where you store shapshot data)  
Insert a config record about your mysql instacne,just like:  
INSERT INTO `myawr\_host`(id,host\_name,ip\_addr,port,db\_role,version, running\_thread\_threshold,times\_per\_hour) VALUES (6, 'db2.11', '192.168.2.11', 3306, 'master', '5.5.27',10000,0);   
Running\_thread\_threshold and times\_per\_hour control whether collect peak time information or not.  
Running\_thread\_threshold is a trigger for status Threads\_running.  
Times\_per\_hour control the times of collection in lasted a hour.  
If you want to collect peak time infomation ,They have to :   
running\_thread\_threshold<=now\_running\_threads and times\_saved   
3.3 add two jobs in crontab(That mysql instance you want to watch,perl-DBD-MySQL is required)   
grant all on **.** to 'superuser'@'localhost' identified by "111111";   
\* \* \* \* \* perl /data/mysql/sh/myawr.pl -u user -p 111111 -lh 192.168.2.11 -P 3306 -tu user -tp 111111 -TP 3306 -th 192.168.1.92 -n eth0 -d sdb1 -I 6 >> /data/mysql/sh/myawr\_pl.log 2>&1  
#  
15 14 \* \* \* /data/mysql/sh/pt-query-digest -~~user=user --password=111111 --review h=192.168.1.92,D=myawr,t=myawr\_query\_review --review-history h=192.168.1.92,D=myawr,t=myawr\_query\_review\_history --no-report --limit=100% --filter="\$event~~>{add\_column} = length(\$event->{arg}) and \$event->{hostid}=6" /data/mysql/sh/slow\_`date -d "-1 day" +"%Y%m%d"`.log >> /data/mysql/sh/pt-query\_run.log 2>&1   
myawr.pl Parameters:  
~~h,~~-help Print Help Info.   
~~i,~~-interval Time(second) Interval(default 1).   
~~d,~~-disk Disk Info(can't be null,default sda1).  
~~n,~~-net Net Info(default eth0).  
~~P,~~-port Port number to use for local mysql connection(default 3306).  
~~u,~~-user user name for local mysql(default user).  
~~p,~~-pswd user password for local mysql(can't be null).  
~~lh,~~-lhost localhost(ip) for mysql where info is got(can't be null).  
~~TP,~~-tport Port number to use formysql where info is saved (default 3306)  
~~tu,~~-tuser user name for mysql where info is saved(default user).  
~~tp,~~-pswd user password for mysql where info is saved(can't be null).  
~~th,~~-thost host(ip) for mysql where info is saved(can't be null).  
~~I,~~-tid db instance register id(can't be null,Reference myawr\_host.id)   
pt-query-digest Parameters:  
--user user name for mysql where info is saved  
--password user password for mysql where info is saved  
--review Store a sample of each class of query in this DSN  
h host(ip) for mysql where info is saved  
D database  
t table name  
--review-historyThe table in which to store historical values for review trend analysis.  
h host(ip) for mysql where info is saved  
D database  
t table name  
$event->{hostid}=6db instance register id(Reference myawr\_host.id)   
The pt-query-digest only support mechanism for switching a slow log file every day just now, named like slow\_20130521.log(slow\_date -d "-1 day" +"%Y%m%d".log)

#### ****Dependencies****

perl-DBD-mysql  
you can install it two way:  
yum install perl-DBD-MySQL  
or install manually like :  
mkdir /tmp/mysqldbd-install   
cp /usr/lib64/mysql/\*.a /tmp/mysqldbd-install  
perl Makefile.PL --libs="-L/tmp/mysqldbd-install -lmysqlclient"   
make   
make test   
make install

#### ****Mysql WorkLoad Report****

We can use myawrrpt.pl to generate mysql workload report.You can execute the script on MySQL database machine where store the performance data,but perl-DBD-MySQL is required.We also can execute the script in any linux machine with perl-DBD-MySQL installed.   
You can execute it for help Info:  
perl myawrrpt.pl -h   
Info :  
Created By noodba (www.noodba.com).  
References: Oracle awr  
Usage :  
Command line options :   
~~h,~~-help Print Help Info.   
~~P,~~-port Port number to use for local mysql connection(default 3306).  
~~u,~~-user user name for local mysql(default user).  
~~p,~~-pswd user password for local mysql(can't be null).  
~~lh,~~-lhost localhost(ip) for mysql where info is got(can't be null).   
~~I,~~-tid db instance register id(can't be null,Ref myawr\_host.id)   
  
Sample :  
shell> perl myawrrpt.pl -p 111111 -lh 192.168.1.111 -I 11  
============================================================================   
Let use to generate mysql (db2.11,instance id = 6) workload report:  
perl myawrrpt.pl -u user -p 111111 -P 3306 -lh 192.168.1.92 -I 6  
===================================================  
| Welcome to use the myawrrpt tool !   
| Date: 2013-05-22  
|  
| Hostname is: db2.11   
| Ip addr is: 192.168.2.11   
| Port is: 3306   
| Db role is: master   
|Server version is: 5.5.27  
| Uptime is: 0y 2m 2d 7h 55mi 33s  
|  
| Min snap\_id is: 1   
| Min snap\_time is: 2013-05-21 14:12:02   
| Max snap\_id is: 1147   
| Max snap\_time is: 2013-05-22 09:29:02   
| snap interval is: 60s  
===================================================   
Listing the last 2 days Snapshots  
---------------------------------  
snap\_id: 19 snap\_time : 2013-05-21 14:30:02   
snap\_id: 38 snap\_time : 2013-05-21 14:49:02   
snap\_id: 57 snap\_time : 2013-05-21 15:08:02   
snap\_id: 76 snap\_time : 2013-05-21 15:27:02   
snap\_id: 95 snap\_time : 2013-05-21 15:46:02   
.....................................................   
snap\_id: 1102 snap\_time : 2013-05-22 08:44:02   
snap\_id: 1121 snap\_time : 2013-05-22 09:03:02   
snap\_id: 1140 snap\_time : 2013-05-22 09:22:02   
snap\_id: 1147 snap\_time : 2013-05-22 09:29:02   
Pls select Start and End Snapshot Id  
------------------------------------  
Enter value for start\_snap:1  
Start Snapshot Id Is:1   
Enter value for end\_snap:589  
End Snapshot Id Is:589   
Set the Report Name  
-------------------   
Enter value for report\_name:myawr.html   
Using the report name :myawr.html   
Generating the mysql report for this analysis ...  
Generate the mysql report Successfully.   
[mysql@test2 myawr]$ ls -al  
total 976  
drwxrwxr-x 2 mysql mysql 4096 May 22 09:30 .  
drwx------ 19 mysql mysql 4096 May 13 10:42 ..  
~~rw-rw-r~~- 1 mysql mysql 73074 May 22 09:30 myawr.html  
~~rw-rw-r~~- 1 mysql mysql 53621 May 11 16:23 myawrrpt.pl   
Mysql WorkLoad Report   
Ok,let me show some pictures which come from my test db report:

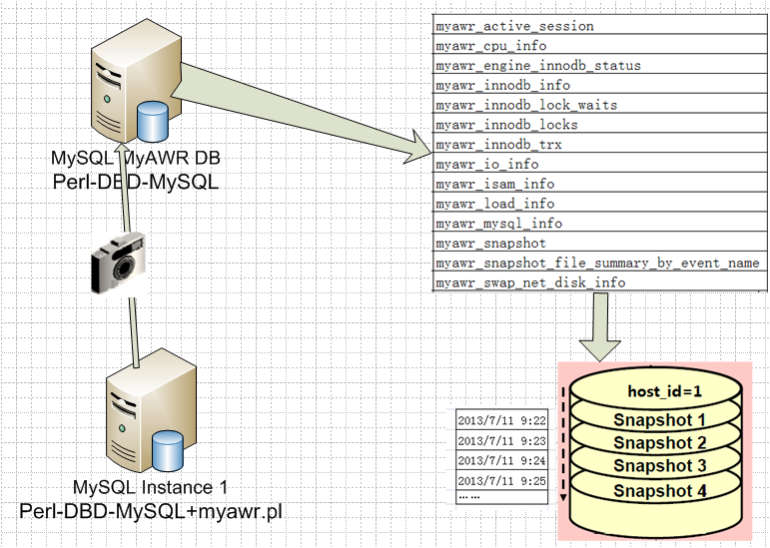
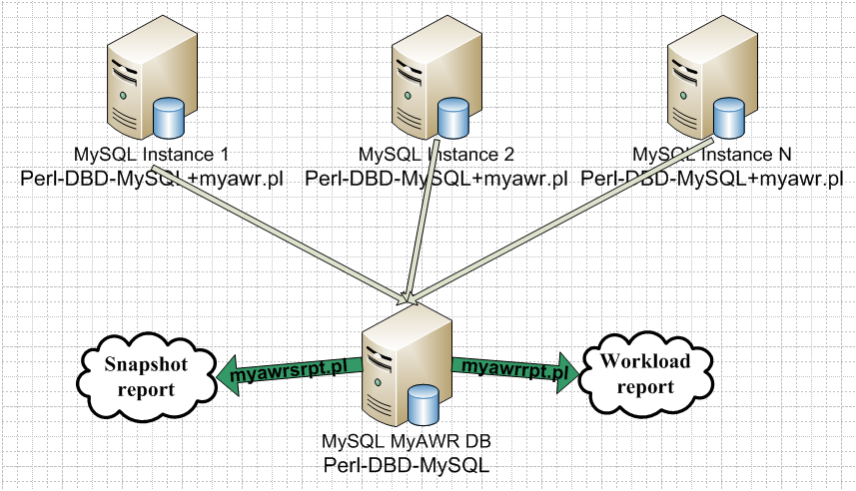
For detail report ,pls click [myawr.html](http://www.noodba.com/myawr.html)

#### ****Mysql Snapshot Report****

We can use myawrsrpt.pl to generate mysql snapshot report. You can execute the script on MySQL database machine where store the performance data,but perl-DBD-MySQL is required.We also can execute the script in any linux machine with perl-DBD-MySQL installed.   
You can execute it for help Info:  
[root@oel58 myawr2.0]# perl myawrsrpt.pl   
==========================================================================================  
Info :  
Created By noodba (www.noodba.com).   
Usage :  
Command line options :   
~~h,~~-help Print Help Info.   
~~P,~~-port Port number to use for local mysql connection(default 3306).  
~~u,~~-user user name for local mysql(default user).  
~~p,~~-pswd user password for local mysql(can't be null).  
~~lh,~~-lhost localhost(ip) for mysql where info is got(can't be null).  
~~n,~~-rname file name for snapshot report(default "snaprpt\_" + "hostid" + "\_" + "snapid" + ".html").  
~~s,~~-snapid snap id(can't be null).   
~~I,~~-tid db instance register id(can't be null).   
Sample :  
shell> perl myawrrpt.pl -p 111111 -lh 192.168.1.111 -I 11 -s 26  
==========================================================================================   
Let use to generate mysql (db2.11,instance id = 6) snapshot report:   
[root@oel58 myawr2.0]# perl myawrsrpt.pl -p 123456 -lh 192.168.137.4 -u qwsh -P 3306 -I 6 -s 27  
===================================================  
| Welcome to use the myawrrpt tool !   
| Date: 2013-06-18  
|  
| Hostname is: db2.11   
| Ip addr is: 192.168.2.11   
| Port is: 3306   
| Db role is: master   
|Server version is: 5.5.27  
| Uptime is: 0y 0m 1d 1h 10mi 12s  
===================================================   
Generating the mysql report for this analysis ...  
Generate the mysql report Successfully.  
For detail report ,pls click [myawr\_snapshot.html](http://www.noodba.com/myawr_snapshot.html)

#### ****MySQL AWR 部署和报告生成****

怎么快速部署一套myawr，最少资源配置：linux server一台，DBI，DBD，如果没有mysql的话也要mysql安装包。  
下面介绍一下怎么部署，这个环境是Oracle Linux Server release 5.8，没有mysql，同时准备把myawr的采集对象和存储数据都放在这台上面，正常情况下采集对象上只需安装perl-dbd-mysql，然后部署监控脚本就行了，他只对所监控的mysql数据库进行查询，不会修改任何东西：  
Myawr的架构图:   
我的资源列表  
[root@oel58 myawr]# ls -al  
total 75848  
drwxr-xr-x 2 root root 4096 Jul 7 17:01 .  
drwxr-x--- 18 root root 4096 Jul 7 17:01 ..  
~~rw-rr~~- 1 root root 133427 Jul 7 17:01 DBD-mysql-4.018.tar.gz  
~~rw-rr~~- 1 root root 571379 Jul 7 17:01 DBI-1.615.tar.gz  
~~rw-rr~~- 1 root root 32485 Jul 7 17:01 myawr.pl  
~~rw-rr~~- 1 root root 64206 Jul 7 17:01 myawrrpt.pl  
~~rw-rr~~- 1 root root 35706 Jul 7 17:01 myawrsrpt.pl  
~~rw-rr~~- 1 root root 17899890 Jul 7 17:01 MySQL-client-5.5.29-2.rhel5.i386.rpm  
~~rw-rr~~- 1 root root 3606645 Jul 7 17:01 MySQL-devel-5.5.29-2.rhel5.i386.rpm  
~~rw-rr~~- 1 root root 54638727 Jul 7 17:01 MySQL-server-5.5.29-2.rhel5.i386.rpm  
~~rw-rr~~- 1 root root 549438 Jul 7 17:01 pt-query-digest  
以下是安装步骤(下面的操作我是在同一个虚拟机上进行的操作)：



**1.环境检查：**

[root@oel58 ~]# rpm -qa | grep -i mysql

[root@oel58 ~]#

[root@oel58 ~]# rpm -qa | grep -i dbd

[root@oel58 ~]#

[root@oel58 ~]# rpm -qa | grep -i dbi

[root@oel58 ~]#

[root@oel58 ~]# perl -version

This is perl, v5.8.8 built for i386-linux-thread-multi

Copyright 1987-2006, Larry Wall

[root@oel58 ~]# uname -a

Linux oel58 2.6.32-300.10.1.el5uek #1 SMP Wed Feb 22 17:22:40 EST 2012 i686 i686 i386 GNU/Linux

[root@oel58 ~]#

[root@oel58 ~]#

[root@oel58 ~]# getconf LONG\_BIT

32

**2.安装mysql**

最简单的安装法就是使用rpm安装，这个可以在   
<http://downloads.mysql.com/archives.php?p=mysql-5.5>上下载。   
[root@oel58 ~]# rpm -ivh MySQL-server-5.5.29-2.rhel5.i386.rpm  
Preparing... ########################################### [100%]  
1:MySQL-server ########################################### [100%]   
  
[root@oel58 ~]#   
[root@oel58 ~]#   
[root@oel58 ~]# rpm -ivh MySQL-client-5.5.29-2.rhel5.i386.rpm  
Preparing... ########################################### [100%]  
1:MySQL-client ########################################### [100%]   
[root@oel58 ~]# rpm -ivh MySQL-devel-5.5.29-2.rhel5.i386.rpm  
Preparing... ########################################### [100%]  
1:MySQL-devel ########################################### [100%]  
[root@oel58 ~]# service mysql start  
Starting MySQL... [ OK ]  
[root@oel58 ~]# /usr/bin/mysqladmin -u root password '123456'   
  
[root@oel58 ~]# cat /etc/my.cnf   
[mysqld]  
performance\_schema  
log\_bin=on  
innodb\_file\_per\_table=1  
slow\_query\_log=1  
long\_query\_time=1  
log\_queries\_not\_using\_indexes=0   
  
MySQL 5.5新增PERFORMANCE\_SCHEMA ,主要用于收集数据库服务器性能参数。包括等待的信息，事件汇总信息等。  
myawr不是强制要求开启PERFORMANCE\_SCHEMA的，开启的话，他会采集一些数据，在展示时会有top events信息。

**3.安装mysql dbd(依赖dbi)**

cd DBI-1.615  
perl Makefile.PL   
make  
make install   
mkdir /tmp/mysqldbd-install   
cp /usr/lib/mysql/\*.a /tmp/mysqldbd-install  
cd DBD-mysql-4.018  
perl Makefile.PL --libs="-L/tmp/mysqldbd-install -lmysqlclient"   
make   
make test   
make install   
  
检查安装模块：   
[root@oel58 aaa]# cat listmodle.pl   
#!usr/bin/perl  
use strict;  
use warnings;   
use ExtUtils::Installed;   
  
my $inst = ExtUtils::Installed->new();  
print join "\n", $inst->modules();   
  
[root@oel58 ~]# perl listmodle.pl   
DBD::mysql  
DBI  
Perl   
  
检查DBD的可用性：   
CREATE TABLE `test` (  
`name` varchar(50) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;   
INSERT INTO test (name) VALUES ('hello world');   
  
[root@oel58 ~]# cat dbd.pl   
#!/usr/bin/perl   
use DBI;   
my $dbh = DBI->connect( "DBI:mysql:database=test;host=localhost",  
"root", "123456", { 'RaiseError' => 1 } );  
#my $rows = $dbh->do("INSERT INTO test (name) VALUES ('hello world')");  
my $query = $dbh->prepare("SELECT name FROM test");  
$query->execute();   
while ( $ary = $query->fetchrow\_arrayref() ) {  
print ( $ary->[0] . "\n");  
}   
  
如果mysql是自己编译安装或者二进制安装的，可以直接：  
下面这个因为我是rpm安装的，有冲突   
[root@oel58 ~]# yum install perl-DBD-MySQL  
Loaded plugins: rhnplugin, security  
This system is not registered with ULN.  
ULN support will be disabled.  
Setting up Install Process  
Resolving Dependencies  
--> Running transaction check  
---> Package perl-DBD-MySQL.i386 0:3.0007-2.el5 set to be updated  
--> Processing Dependency: perl(DBI) for package: perl-DBD-MySQL  
--> Processing Dependency: libmysqlclient.so.15(libmysqlclient\_15) for package: perl-DBD-MySQL  
--> Processing Dependency: libmysqlclient.so.15 for package: perl-DBD-MySQL  
--> Running transaction check  
---> Package mysql.i386 0:5.0.95-1.el5\_7.1 set to be updated  
---> Package perl-DBI.i386 0:1.52-2.el5 set to be updated  
--> Processing Conflict: MySQL-server conflicts mysql  
--> Processing Conflict: MySQL-client conflicts mysql  
--> Processing Conflict: mysql conflicts MySQL  
--> Finished Dependency Resolution  
MySQL-client-5.5.29-2.rhel5.i386 from installed has depsolving problems  
--> MySQL-client conflicts with mysql  
mysql-5.0.95-1.el5\_7.1.i386 from el5\_latest has depsolving problems  
--> mysql conflicts with MySQL-server  
MySQL-server-5.5.29-2.rhel5.i386 from installed has depsolving problems  
--> MySQL-server conflicts with mysql  
Error: MySQL-server conflicts with mysql  
Error: mysql conflicts with MySQL-server  
Error: MySQL-client conflicts with mysql  
You could try using --skip-broken to work around the problem  
You could try running: package-cleanup --problems  
package-cleanup --dupes  
rpm -Va --nofiles --nodigest  
The program package-cleanup is found in the yum-utils package.

**4.初始化数据库(注意：这个是在你要保存信息的mysql数据库上部署)**

执行myawr.sql  
增加配置信息：  
INSERT INTO `myawr\_host`(id,host\_name,ip\_addr,port,db\_role,version, running\_thread\_threshold,times\_per\_hour) VALUES (6, 'db2.11', '192.168.2.11', 3306, 'master', '5.5.27',10000,0);

**5. job(注意：这个是在你想要采集信息的mysql数据库上部署)：**

grant all on **.** to 'superuser'@'localhost' identified by "111111";   
在把下列脚本部署成job前，可以先手工运行一下：   
\* \* \* \* \* perl /root/myawr/myawr.pl -u superuser -p 111111 -lh localhost -P 3306 -tu myuser -tp 111111 -TP 3306 -th 192.168.137.9 -n eth0 -d sda3 -I 6 >> /root/myawr/myawr\_pl.log 2>&1   
  
下面的脚本可以模拟有lock wait的情况：  
use test;  
create table txtest  
(id int primary key ,  
name varchar(200))  
engine=innodb default charset = utf8 ;   
insert into txtest values(1, 'xxxxxxxxxxxxxxxxxxxx');  
insert into txtest values(2, 'zzzzzzzzzzzzzzzzzzzzzzz');  
insert into txtest values(3, 'sssssssssssssssssssss');   
commit;   
  
###事物一  
start transaction;  
update txtest set name='ddddddddd' where id =1 ;   
update txtest set name='wwwwwwwwwwwddddddddd' where id =2 ;   
###事物二  
start transaction;  
update txtest set name='kkkkkkkkkkkkkkkkkk' where id =1 ;   
  
###事物二  
select sleep(300);   
  
在把下列脚本部署成job前，可以先手工运行一下：  
./pt-query-digest -~~user=myuser --password=111111 --review h=192.168.137.9,D=myawr,t=myawr\_query\_review --review-history h=192.168.137.9,D=myawr,t=myawr\_query\_review\_history --no-report --limit=100% --filter=" \$event~~>{add\_column} = length(\$event->{arg}) and \$event->{hostid}=6" /var/lib/mysql/oel58-slow.log

**6.生成report：**

perl myawrrpt.pl -u myuser -p 111111 -P 3306 -lh 192.168.137.9 -I 6  
perl myawrsrpt.pl -u myuser -p 111111 -P 3306 -lh 192.168.137.9 -I 6 -s 29