51testing
博为峰软件

软件性能测试

云层

现在遇到的问题

- 不知道怎么做性能测试？
- 到底性能测试做了能干啥？
- 看不懂性能测试报告？
- 知道有问题就是定位不到？
- 性能需求分析不出来？
- 脚本不会写怎么办？
- 要会的东西太多不知道学啥？
- 貌似没啥可以学的了？

我们遇到的性能问题1

12306铁路客户服务网站，作为铁道部唯一指定官方火车售票网站，一直承受巨大并发访问量。统计显示，12306铁路客户服务网站，自1月5日起，连续5天日均点击数超过10亿次。仅1月9号一天，12306.cn的日点击量突破了14亿，这相当于平均全中国每个人都点击了一次。

面对“世界第一网站”，铁道部部长盛光祖之言，现在12306订票网站一天的点击量超过了14亿,相当于全中国每个人都点击了一次,可以说是“世界第一网站”,但这个“世界第一”要不得!

这个“要不得”意味深长。一来说明铁路运力相对春运客流还是明显滞后，“一票难求”的局面远没有解决。二来铁道部长的如此低调行事，恐怕与备受指责甚至遭受谩骂的12306网站频繁瘫痪脱不了干系。

我们看一下铁道部最近公布的官方信息。中国铁道部1月13日披露，新一代客票系统的规划和设计已经启动。包括即将引入云计算技术，以科学成熟的体系架构为基础，构建支撑超大规模并发交易、海量数据存储、灵活扩展、兼容性良好、安全可靠高效的综合信息系统。

可见，铁道部已经承认12306网站采用了一个不够科学，尚未成熟的架构。

我们遇到的性能问题2

- **Oracle Exadata 简介**

  随着企业业务的发展，大型数据仓库越来越多，其规模也在迅速扩大，平均每两年规模增大3倍。大型数据仓库要求以最高的磁盘读取速度扫描几十、几百或几千个磁盘，只有磁盘和服务器之间的管道带宽增加10倍或更多才能满足此要求，所以企业常常发现数据仓库越大，运行速度可能就越慢。

  如何突破数据带宽瓶颈？一个全新的架构---Oracle Exadata[1]应运而生。甲骨文公司首席执行官Larry Ellison 和Sun Microsystems 公司执行副总裁John Fowler，在2009年9月宣布：推出世界上第一个OLTP数据库机——Sun Oracle数据库机（即Oracle Exadata第二版）。

  “Exadata数据库机将成为甲骨文30年发展史中最成功的新产品，”Larry Ellison曾经这样对这款产品给予厚望。Oracle的Exadata 第二版是Sun硬件与Oracle数据库软件的结合体，而第一版是由Oracle和HP联合打造的。

  Sun Oracle数据库机采用业界标准硬件组件以及Sun公司的FlashFire技术、Oracle数据库11g第二版（Oracle Database 11g Release 2）和Oracle Exadata存储服务器软件11.2版（Oracle Exadata Storage Server Software Release 11.2），在用于数据仓库时，其运行速度是第一版的两倍。

我们遇到的性能问题3

- 甲骨文公司今天在甲骨文全球技术与应用大会（Oracle OpenWorld）上宣布，为优化Oracle软件和硬件而推出一款快速、现代和可靠的Linux内核——Oracle Unbreakable企业级内核（Unbreakable Enterprise Kernel）。

  Oracle Unbreakable企业级内核是Oracle Linux的一部分，其原名为Oracle企业级Linux。

  在Oracle Linux、数据库、中间件和硬件工程设计团队的共同努力下，Oracle Unbreakable企业级内核具有如下优势：

  快速：OLTP性能测试显示其性能增强超过Red Hat兼容内核75%，Infiniband信息传递速度提升了200%; 固态磁盘访问速度提升了137%。

  现代：为大型NUMA服务器提供了优化;改进了电能管理和能率，精炼了CPU和内存资源控制。

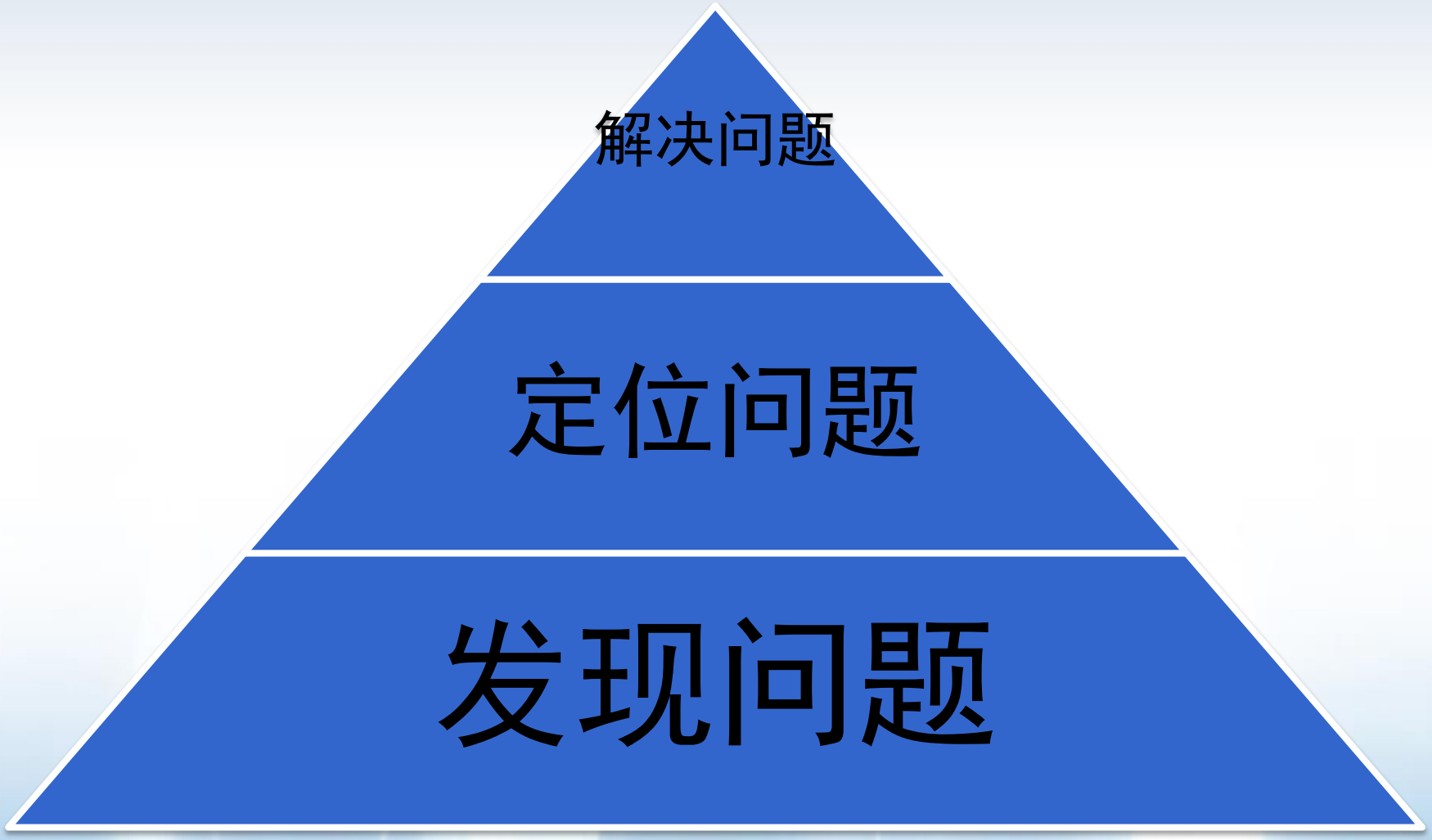
  可靠：支持数据集成扩展和T10保护信息模式，防止数据在写入存储过程中被损坏；硬件故障管理能提高应用的正常运行时间；降低性能跟踪的计数器使用成本。

  面向Oracle的最优化：通过创建和测试来运行Oracle硬件、数据库和中间件，获得了迄今为止最佳的Linux性能与可靠性。

  最新版本的Oracle Exadata数据库机和新Oracle Exalogic Elastic Cloud借助Oracle Unbreakable企业级内核实现了最极致的Linux性能。

那么性能问题有哪些呢？

我们的职责是什么？



如何应对这种问题？

- 何为测试？

  软件测试就是利用测试工具按照测试方案和流程对产品进行功能和性能测试，甚至根据需要编写不同的测试工具，设计和维护测试系统，对测试方案可能出现的问题进行分析和评估。执行测试用例后，需要跟踪故障，以确保开发的产品适合需求。

- 何为性能测试

  就是通过测试验证系统能够满足需求。

- 我们纠结在什么地方？

  错误的理解了这个职位的定义。

- 为什么就算是这样你还会纠结于此？

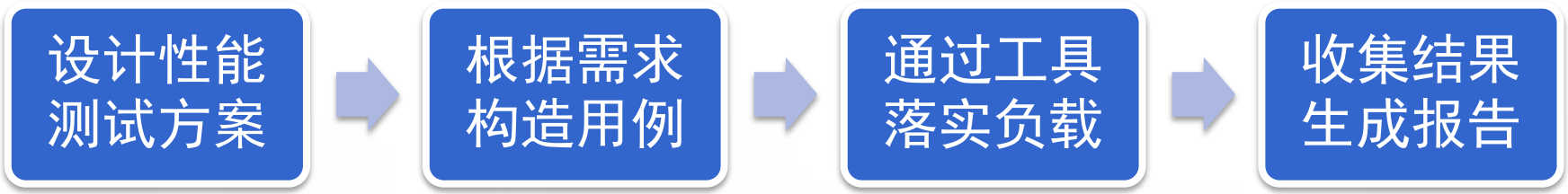
  国情！

性能测试的实质

- 性能测试 -- Performance Testing：
  - 在一定的负载情况下，系统的响应时间等特性是否满足特定的性能需求，较广义的一个概念
  - 通常收集所有和测试有关的所有性能以建立Benchmark, 所得数据被不同人在不同场合下进行使用
  - 性能测试的目的不是去找Bug, 而是排除系统的瓶颈
  - 关注点：How much 和 How fast

- 区分以下用户数概念：
  - 在线用户数：所有正在访问系统用户(不一定作操作)
  - 并发用户数：同时对服务器产生请求的用户总数
  - 系统用户数：系统额定的用户数量(设计容量)

我们该做什么？



太天真了！

拿着卖白菜的钱，操着卖白粉的心

面对现实怎么办？

- 将遇良才？
- 个人英雄？
- 团队合作？
- 一专多能？
- 趁早转行？

既然现实如此那么我们就来见招拆招吧！

Step List

1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*

客户交流

- 响应时间
- 处理能力
- 资源占用率
- 负载量

历史日志

- 最大峰值访问量
- 负载特性
- 行业规则

同类产品

- 物理特征
- 最大负载量
- 系统性能状态

80/20

- 核心业务
- 核心负载时间
- 经验

客户交流案例

- Xxx企业需要上一套系统，但不知是否会存在性能问题，故咨询解决方案。
  - 客户懂性能？
  - 客户不懂性能？
  - 客户真的懂性能么？
- 引导客户明确性能测试目标是解决问题的根本。

我家孩子学习成绩很好，我想让他跳级，但是不知道会不会跟不上？

历史日志案例

- XXX公司需要新增一套流程管理系统，但不知道该系统在什么时候会出现性能问题？

2012年日最大发起待办94598笔（含发起流程），假定在1小时内完成，每笔完成时间为5秒，则并发量为：94598*5/3600 =131。

以每年20%的数据递增估算，5年后流程数量并发数：131 * 1.2 * 1.2 * 1.2 * 1.2 * 1.2 = 325

按照收入规模估算，广东5年后流程数量并发数预计为325 * 1.5 = 488

按照收入规模估算，江苏、上海合并部署，5年后流程数量并发数预计为325 * 1.8 = 585

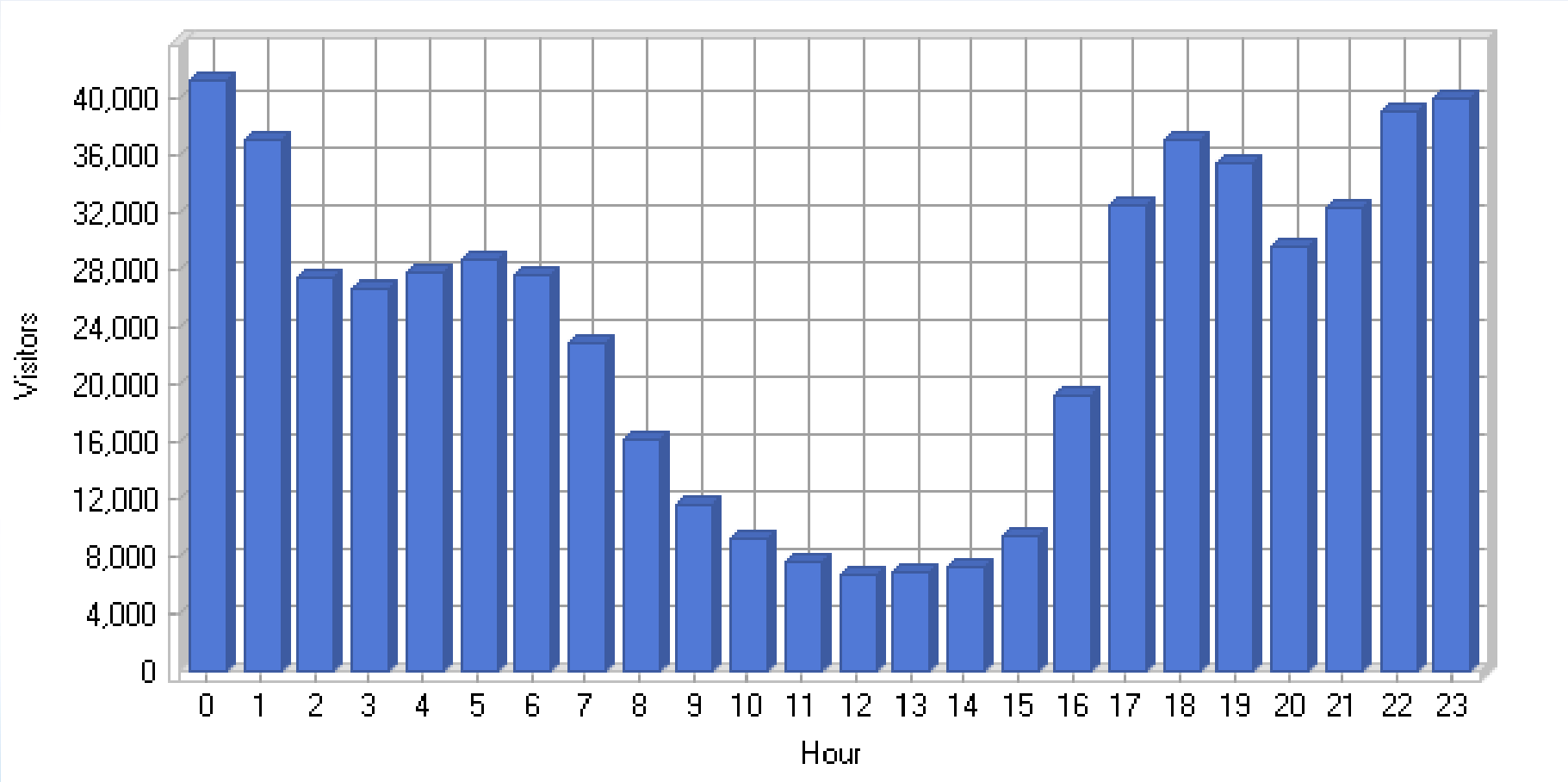
日志分析

Time range: 2008/8/19 22:15:24 - 2008/9/1 20:51:53	Generated on Mon Sep 22, 2008 - 11:59:35
Summary	
Hits	
Total Hits	29,930,396
Average Hits per Day	2,137,885
Average Hits per Visitor	51.48
Cached Requests	13,244,961
Failed Requests	754,898
Page Views	
Total Page Views	6,340,382
Average Page Views per Day	452,884
Average Page Views per Visitor	10.90
Visitors	
Total Visitors	581,427
Average Visitors per Day	41,530
Total Unique Ips	315,632
Bandwidth	
Total Bandwidth	1849.27 GB
Average Bandwidth per Day	132.09 GB
Average Bandwidth per Hit	64.79 KB
Average Bandwidth per Visitor	3.26 MB

日志分析2

Date	Hits	Page Views	Visitors	Average Visit Length	Bandwidth (KB)
Tue 2008/8/19	434,414	52,745	7,006	35:14	20,344,842
Wed 2008/8/20	2,748,087	518,739	53,738	14:49	234,160,955
Thu 2008/8/21	2,823,361	521,378	55,263	14:36	122,409,440
Fri 2008/8/22	2,913,660	651,944	54,998	15:56	140,089,499
Sat 2008/8/23	1,817,904	532,110	46,947	14:30	94,513,643
Sun 2008/8/24	1,261,078	346,792	42,741	15:32	121,878,834
Mon 2008/8/25	2,274,538	482,017	48,886	16:43	84,412,575
Tue 2008/8/26	3,185,667	586,778	47,500	17:53	125,310,525
Wed 2008/8/27	3,032,705	558,642	46,648	16:57	156,203,078
Thu 2008/8/28	2,977,549	540,504	46,360	19:25	144,683,988
Fri 2008/8/29	2,772,113	535,783	46,042	18:45	196,891,020
Sat 2008/8/30	1,853,726	458,623	38,499	19:35	330,775,943
Sun 2008/8/31	1,351,513	424,840	34,931	20:07	110,345,823
Mon 2008/9/1	484,081	129,487	11,868	11:16	57,084,709
Total	29,930,396	6,340,382	581,427	17:00	1,939,104,880

日志分析3


Visitors
40,000
36,000
32,000
28,000
24,000
20,000
16,000
12,000
8,000
4,000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour

如果我要去爬黄山，应该要带多少水？

同类产品案例

- XX银行需要构建构造自己的网上银行，需要对该系统进行性能评估。

- 同事参加了XX的性能测试培训，我也想参加但是不知道好不好？

80/20案例

- XX电信MSS流程引擎系统待上线，不知应该对那些模块进行性能测试，并如何测试？

思考

- 上海世纪大道地铁站的瓶颈在哪里？

需求&分析篇总结

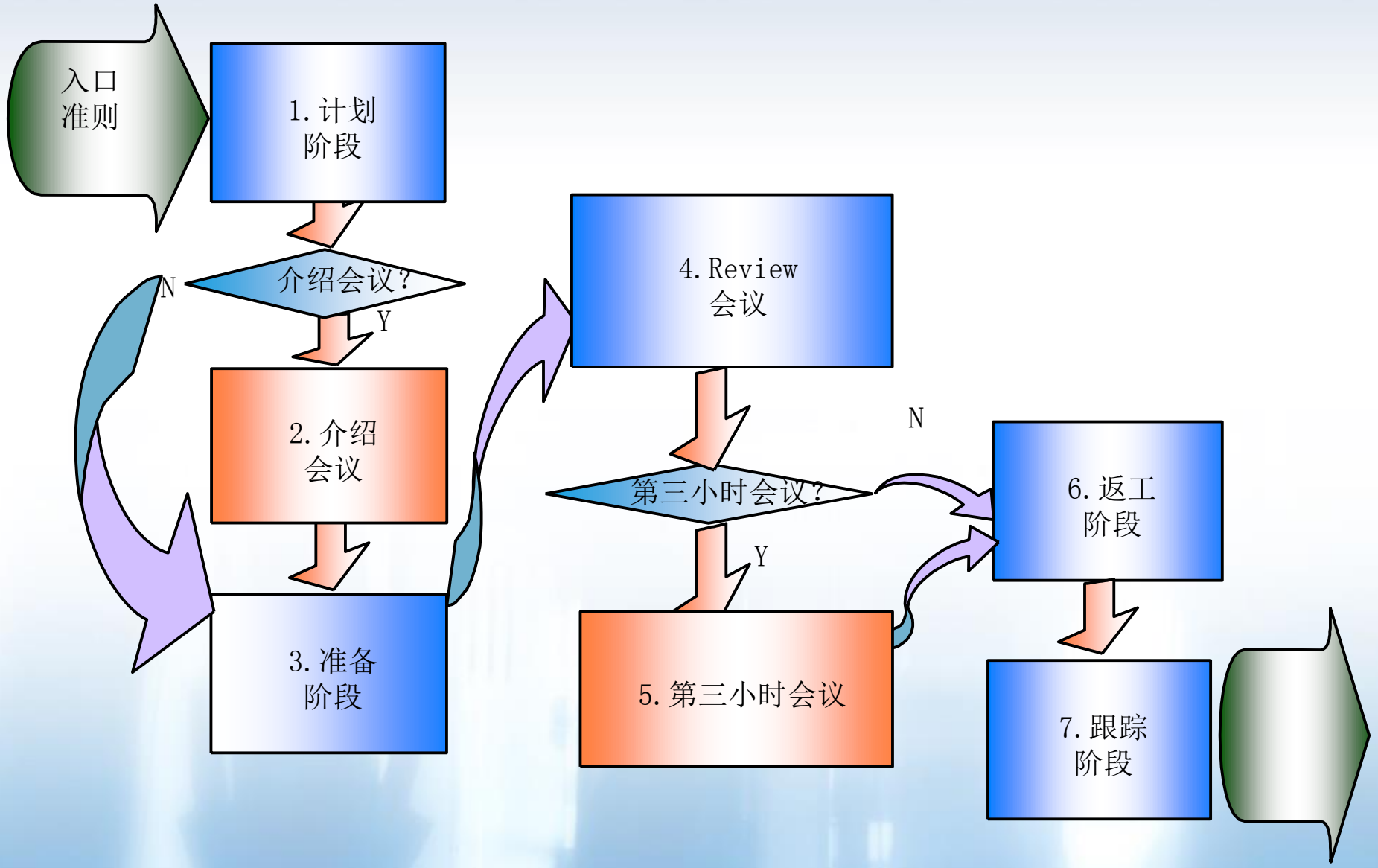
- 性能需求的关键在于
  - 知道测什么
  - 知道被测对象是什么
  - 知道被测对象以后会被怎么用
  - 知道我应该怎么测*
  - 知道我测出来的结果怎么算通过

Step List

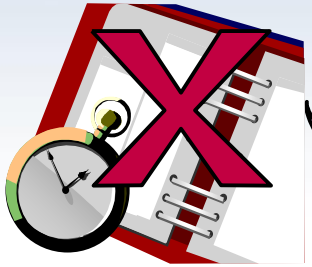
1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*

- 何为评审
  - 为确定主题事项达到规定目标的适宜性、充分性和有效性所进行的活动
- 谁应该参加评审
  - 客户*
    - 架构设计
    - 开发
    - DBA
    - PM
  - 高级性能测试工程师
  - 性能测试工程师

如何做评审


入口
准则
1. 计划
阶段
介绍会议?
N
Y
2. 介绍
会议
3. 准备
阶段
4. Review
会议
第三小时会议?
N
Y
5. 第三小时会议
6. 返工
阶段
7. 跟踪
阶段

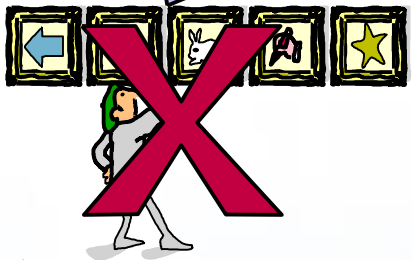
做好评审的关键



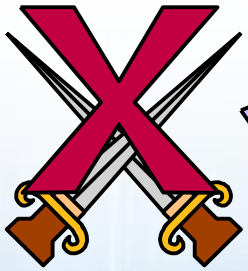
●没有评审计划

●专家选择不合适

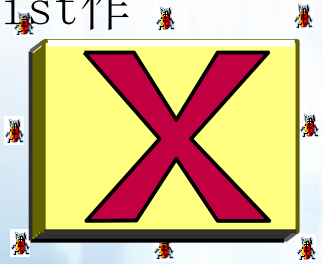
●没有充分的准备



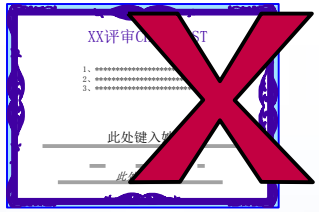
●评审会议偏离主题和重点



●评审会议中过多争论占用大量时间



●问题修改后跟踪不力



●没有使用CheckList作为指导

Step List

1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*

设计篇

- 在知道了测什么的基础上设计什么？
  - 能不能测?
  - 怎么测?
  - 测试计划
  - 测试方案
  - 测试场景
  - 测试流程
  - 测试数据

何为能不能测？

- 爱情不是你想买想买就能买！
- 系统不是你想测想测就能测！
  - 协议
  - 脚本开发
  - 阻碍技术
  - 从整体切入细节或从细节反推整体

怎么测案例

- 某下载软件P2P的性能怎么测？
- 系统将调用西门子某PLC进行机械状态采集，由于采集点众多导致的性能问题？
- 某智能机手机定位服务软件（附近就餐&优惠券）如何进行性能测试？

想测就能测！

- 做到想测就测的关键是分析：
  - 被测对象是一个基于网络的应用还是一个单机应用
  - 被测对象是如何实现负载的
  - 如何伪造数据流
  - 何种工具最适合做这种数据流伪造

性能测试工具崭露头角

- 工具可以帮你简单的实现请求的模拟
- 工具可以帮你简单的实现负载
- 如果没有工具可以自己写一个

计划和方案需要明确的信息

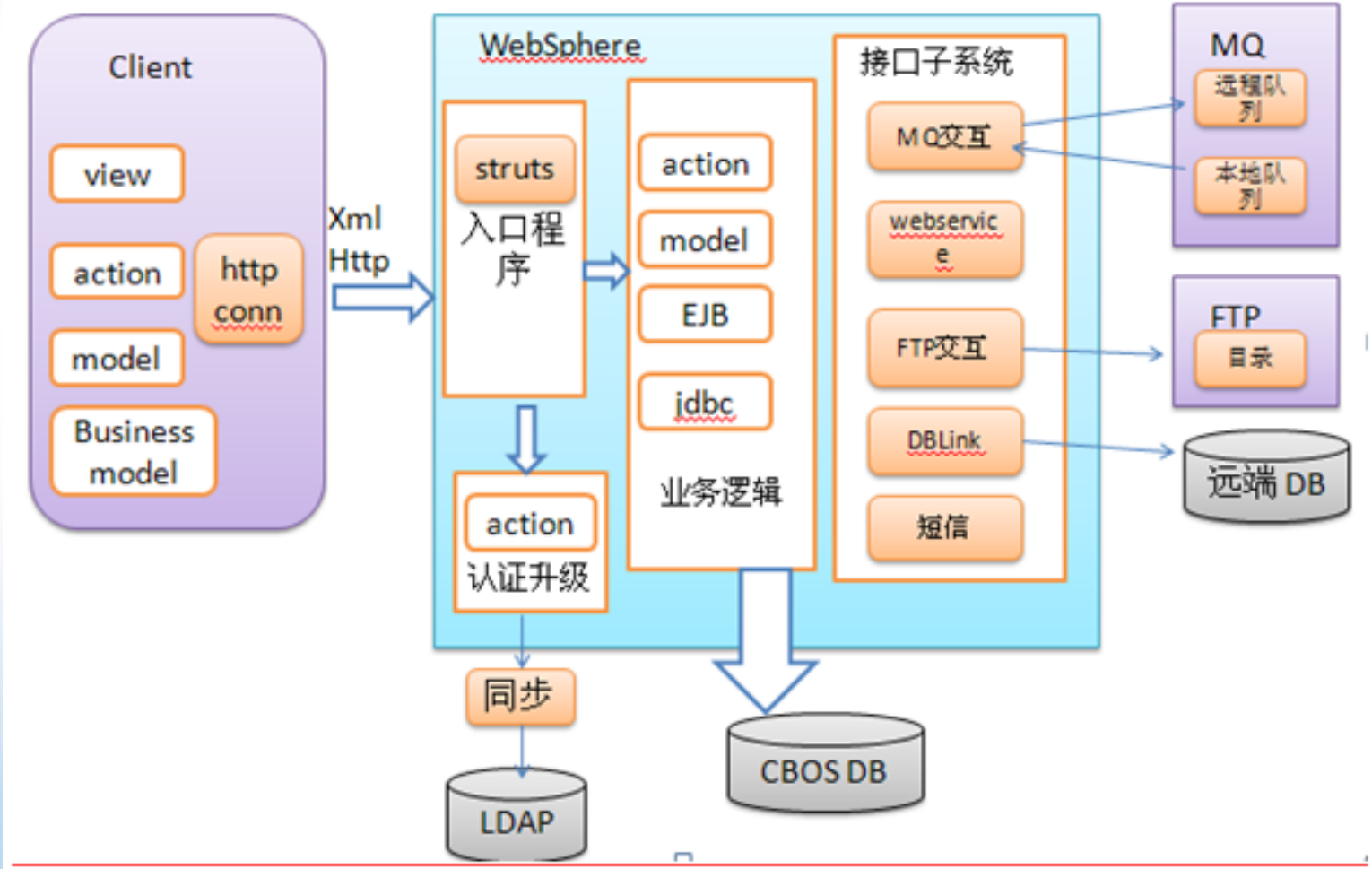
- 系统真实环境平台的架构及软硬件标准
- 需要进行性能测试的模块及对应案例
- 脚本开发中可能需要解决的技术
- 业务完成与否的判断依据
- 相关信息的监控策略
- 场景模型的定义
- 对系统负载后的结果预估

编写测试计划

- 测试计划内容
  - 文档目的
  - 项目背景
  - 相关术语
  - 输入文档
  - 运行环境
  - 测试内容
  - 角色安排
  - 工具及进度安排
  - 输出内容

系统架构

系统逻辑总体架构如下图所示:



编写测试方案

方案与计划的不同在于方案需要指导怎么做

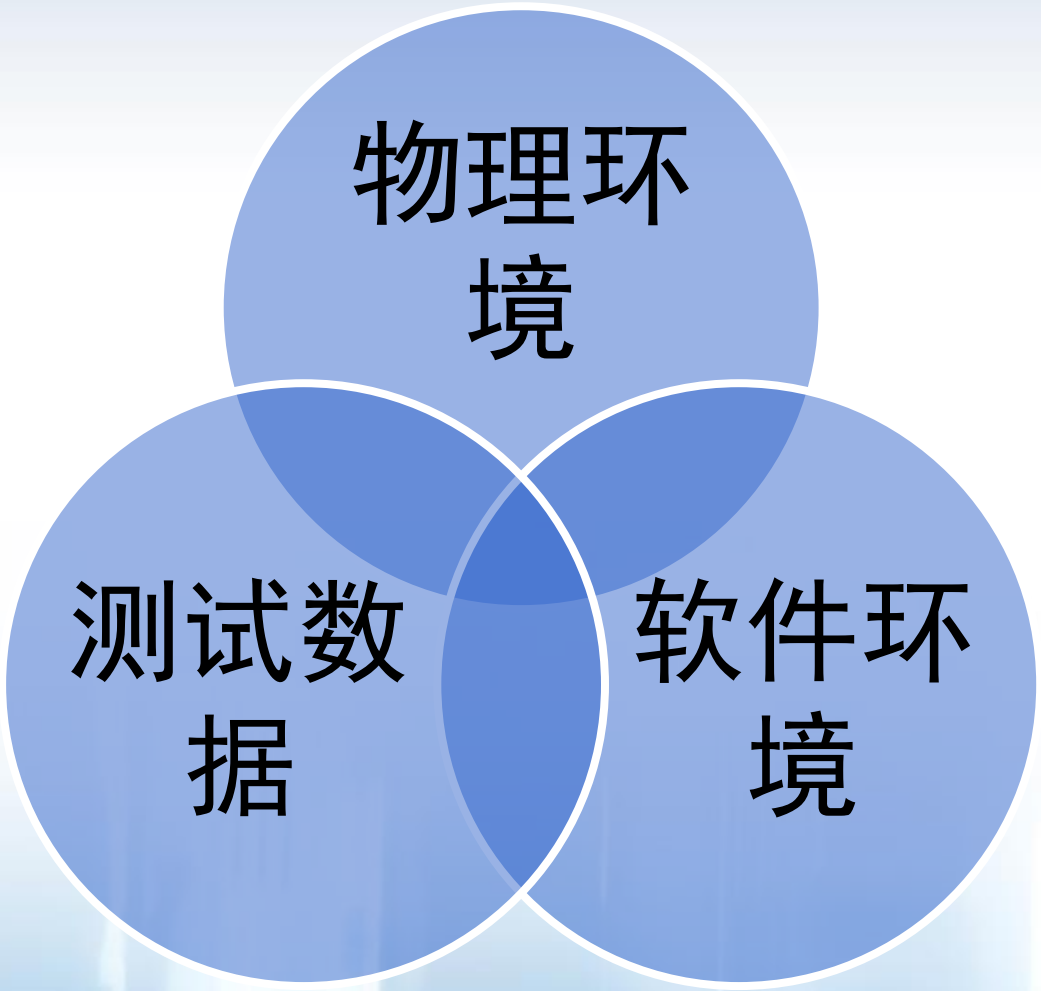
- 测试方案内容
  - 测试目的
  - 测试策略
  - 测试准备流程
  - 测试脚本
  - 负载模型定义
  - 监控定义
  - 场景设定

测试策略

- 每一轮测试的重点
  - 测试内容
  - 性能测试的关注点
  - 确定性能可能的瓶颈
  - 平台对比
  - 稳定性或压力测试

Step List

1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*



环境偏差

- 由于客观原因大多数情况并无法活动和实际情况1:1的测试环境
  - 通过评估建模来解决应用型扩展
  - 通过数学建模来解决物理扩展
- 容量规划的测试评估
  - 表示层和应用层的比例
  - 应用层和数据库层的比例

容量案例

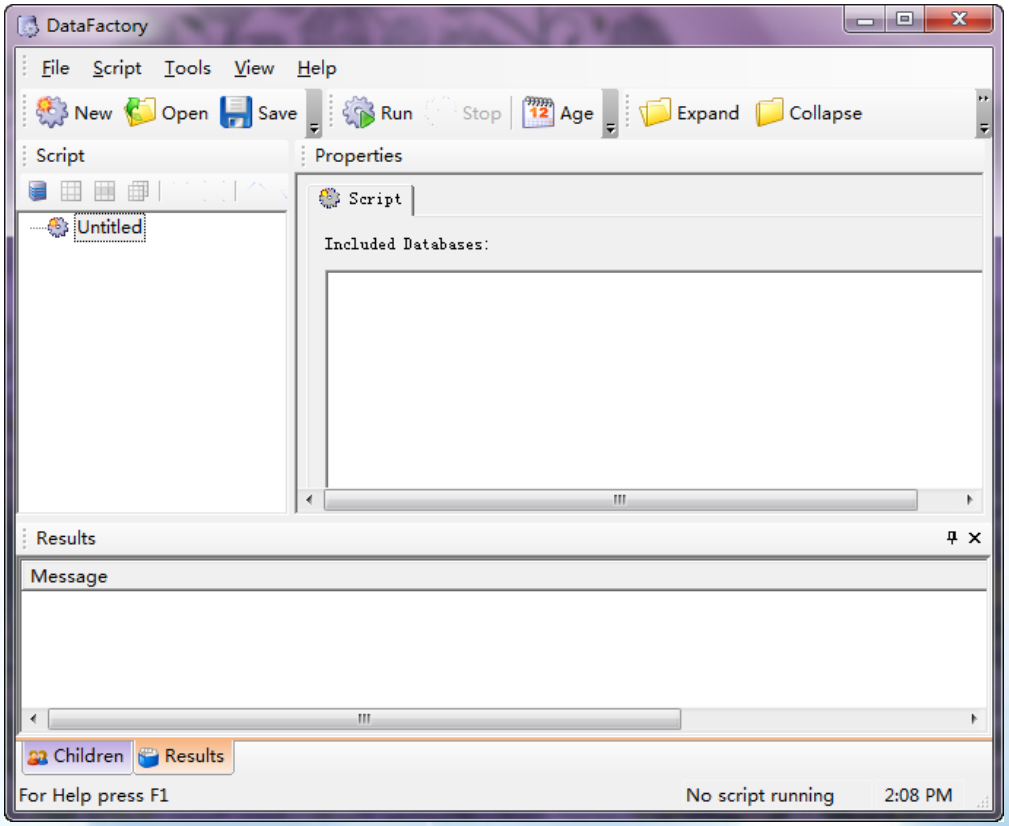
- 在本地环境中某系统现在能接受300用户并发，测试环境为4APP2DB组合，需要评估上线后使用云环境多节点的性能

测试数据

- 测试数据决定了性能测试结果
  - 在需求中需要明确负载下的数据量
  - 为系统构造有效的负载数据
    - 手动
    - 导入
    - 自动（执行，SQL，数据生成工具）

工具生成测试数据

- PowerDesigner
- DataFactory



模拟脚本

```
randstring(int slen)
{
        int i,randid;
        char temp[100]="";
    char character_set[52] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
'N','O', 'P','Q', 'R', 'S', 'T','U', 'V', 'W', 'X','Y', 'Z',
                                                                'a', 'b', 'c', 'd',
'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l','m', 'n', 'o', 'p', 'q', 'r','s', 't', 'u', 'v', 'w', 'x','y','z'};

        for(i=1;i<=slen;i++)
        {
                randid=rand()%52;
                sprintf(temp,"%s%c",temp,character_set[randid]);
        }
        lr_save_string(temp,"postinfo");
}
```

VuGen Script Business Processes

- 脚本说明
  - 便于阐述脚本到底做了什么

设计场景

- 目标型场景
  - 确定需求是否满足
- 手工场景
  - Real world
    - 压力测试
  - Basic
    - 手工调优

监控

- 监控决定了性能测试中能收集到得信息数量
- 监控会降低性能负载
- 监控应该是运维做的事情
- 常用监控工具
  - Nmon
  - Spotlight
  - DynaTrace
  - ANTS
  - Devparter
  - Jprobe

CPU：就像人的大脑，主要负责相关事情的判断以及实际处理的机制。

查询指令：cat /proc/cpuinfo

内存：大脑中的记忆区块，将皮肤、眼睛等所收集到的信息记录起来的地方，以供CPU进行判断。

查询指令：cat /proc/meminfo

硬盘：大脑中的记忆区块，将重要的数据记录起来，以便未来再次使用这些数据。

查询指令：fdisk -l （需要root权限）

Linux监控命令

vmstat

sar

iostat

top

free

uptime

netstat

ps

strace

lsof

iostat –x


```
#iostat –x
Linux 2.6.18-238.el5 (localhost.localdomain)    01/17/2012
avg-cpu:%user  %nice %system %iowait  %steal   %idle
          0.36   0.01    0.65    1.21    0.00   97.78
Device:rrqm/s wrqm/s  r/s  w/s  rsec/s  wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda    26.02  6.80  11.13   7.31  1114.25  112.79  66.55    0.15    8.35   2.66   4.90
sda1   3.03   0.00  0.24    0.01  6.59      0.03    25.87    0.00    2.71   2.61   0.07
sda2   22.93  6.80  10.84  7.29  1106.74   112.76  67.27   0.15    8.46   2.67   4.84
dm-0   0.00   0.00  32.70  14.10 1102.70   112.76  25.97   0.35    7.38   1.02   4.79
dm-1   0.00   0.00  0.41   0.00   3.26      0.00    8.00     0.00    6.55   0.94   0.04
```

在这里列出了当前磁盘的一些基本数据。
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。
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1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）。
如果%util接近100%，说明产生的I/O请求太多，I/O系统已经满负荷，该磁盘可能存在瓶颈。

其中比较重要的参数如下。

%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。

svctm：平均每次设备I/O操作的服务时间。

await：平均每次设备I/O操作的等待时间。

avgqu-sz：平均I/O队列长度。

如果%util接近100%，表明I/O请求太多，I/O系统已经满负荷，磁盘可能存在瓶颈，一般%util大于70%，I/O压力就比较大，读取速度有较多的wait。同时可以结合vmstat查看查看b参数（等待资源的进程数）和wa参数（I/O等待所占用的CPU时间的百分比，高过30%时I/O压力高）。

await 的大小一般取决于服务时间（svctm）及I/O队列的长度和 I/O 请求的发出模式。如果 svctm 比较接近 await，说明 I/O 几乎没有等待时间；如果 await 远大于 svctm，说明 I/O 队列太长，应用得到的响应时间变慢。

如下为形象的比喻。

r/s+w/s 类似于交款人的总数。

平均队列长度（avgqu-sz）类似于单位时间里平均排队人的个数。

平均服务时间（svctm）类似于收银员的收款速度。

平均等待时间（await）类似于平均每人的等待时间。

平均I/O数据（avgrq-sz）类似于平均每人所买的东西多少。

I/O 操作率（%util）类似于收款台前有人排队的时间比例。

设备I/O操作：总I/O(io)/s = r/s（读）+w/s（写）=1.46 + 25.28=26.74。

平均每次设备I/O操作只需要0.36毫秒完成，现在却需要10.57毫秒完成，因为发出的请求太多（每秒26.74个），假如请求是同时发出的，可以这样计算平均等待时间:

平均等待时间=单个I/O服务器时间×(1+2+...+请求总数-1)/请求总数

每秒发出的I/O请求很多，但是平均队列就4，表示这些请求比较均匀，大部分处理还是比较及时的。svctm 一般要小于 await（因为同时等待请求的等待时间被重复计算了），svctm的大小一般与磁盘性能有关，CPU/内存的负荷也会对其有影响，请求过多也会间接导致svctm 的增加。await 的大小一般取决于服务时间（svctm），以及 I/O 队列的长度和 I/O 请求的发出模式。如果 svctm 比较接近 await，说明 I/O 几乎没有等待时间；如果 await 远大于 svctm，说明 I/O 队列太长，应用得到的响应时间变慢，如果响应时间超过了用户限制的范围，这时可以考虑更换更快的磁盘，调整内核 elevator 算法，优化应用，或者升级CPU。

队列长度（avgqu-sz）也可作为衡量系统I/O负荷的指标，但由于 avgqu-sz 是按照单位时间的平均值，所以不能反映瞬间的I/O洪水。

Nmon

nmon -s10 -c60 -f -m /root/

参数解释如下。

-s10 每 10 秒采集一次数据。

-c60 采集 60 次，即为采集10分钟的数据。

-f 生成的数据文件名中包含文件创建的时间。

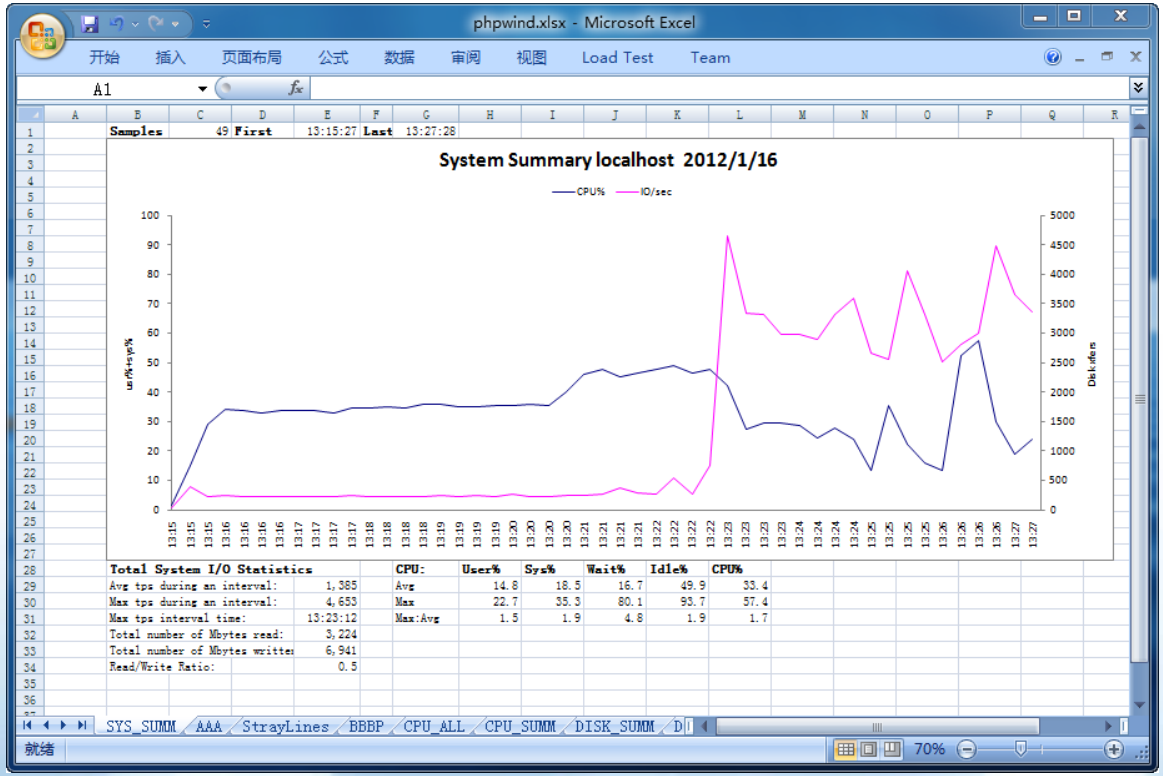
-m 生成的数据文件的存放目录。

这样就会生成一个 nmon 文件，并每10秒更新一次，直到10分钟后。

生成的文件名如： hostname_111221_1316.nmon，hostname是这台主机的主机名。

nmon analyser

- 帮助我们把Nmon数据导出成为图表



- 预测试的重要性
  - 评估系统当前大概性能情况
  - 确定负载是否有效
  - 监控数据是否足够
  - 与预期结果误差范围
- 测试数据的存放及配置管理

场景CheckList

检查项编号	检查项目	检 查 人	备　注
01	场景类型		
02	场景脚本		
03	场景运行设计		
04	运行设置		日志设置 Think time 设置 负载设置 自动化事务设置 带宽设置 地址过滤设置
05	集合点策略		
06	负载生成器		
07	IP虚拟		
08	SLA策略		
09	系统监控及计数器管理		
10	运行结果		
11	环境备份		

场景执行记录

序号	执行时间	测试项目	被测服务器IP	场景结果文件	测试项目数据	备注
CentOS5.6 32位						
1	2011/12/18 12:42	注册用户100	192.168.11.20	res_846.rar	实际注册用户846个	
2	2011/12/18 13:00	查询记录500	192.168.11.20	res.rar		
3	2011/12/18 13:20	随机看帖300	192.168.11.20	res_300.rar		
4	2011/12/18 13:39	随机回帖100	192.168.11.20	res_1326.rar	实际回帖1326	
5	2011/12/18 19:00	注册用户100	192.168.11.21	res_888.rar	实际注册用户888个	
6	2011/12/18 23:40	查询记录500	192.168.11.21	res.rar		300用户开始负载机明显CPU瓶颈
7	2011/12/19 23:41	随机看帖300	192.168.11.21	res_300.rar		400用户负载机CPU瓶颈明显，服务器网络占用一直只有50%

Step List

1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*

- 无论如何尽量不要使用LR或者同类性能测试工具的计数器监控
  - 计数器原理
  - 常见计数器指标

常见计数器

- CPU使用率
- CPU队列长度
- 可用内存数
- 内存硬错误数
- 物理磁盘队列长度
- 物理磁盘读写速度
- 数据库Lock数目
- 连接用户数

CPU计数器

计 数 器	计数器分析
%Processor Time	如果该值持续超过95%，表明瓶颈是CPU。可以考虑增加一个处理器或换一个更快的处理器
Processor Queue Length	指待处理队列中的线程数。即使在有多个处理器的计算机上处理器时间也会有一个单队列。不像磁盘计数器，该计数器仅计算就绪的线程，而不计算运行中的线程。如果在处理器队列中总是有两个以上的线程则通常表示处理器堵塞，处理器瓶颈会导致该值持续大于2。 此外，跟踪计算机的服务器工作队列将显示当前长度的 Server Work Queues\ Queue Length，队列长度持续大于4则表示可能出现处理器拥塞
%User Time	表示耗费CPU的数据库操作，如排序、执行aggregate functions等。如果该值很高，可考虑增加索引，尽量使用简单的表连接、水平分割大表格等方法来降低该值
%Privileged Time	此计数器是特定时间的值，而不是一段时间的平均值。指（CPU内核时间）在特权模式下处理线程执行代码所花时间的百分比。如果该参数的值和“Physical Disk”参数值一直很高，则表明I/O有问题。可考虑更换更快的硬盘系统。另外设置Tempdb in RAM、减低“max async IO”和“max lazy writer IO”等措施都会降低该值
% DPC Time	网卡CPU的占用率，该值越低越好。在多处理器系统中，加入一个网卡可能会提高性能。如果这个值大于50%并且Processor:% Processor Time非常高，说明提供的网络已经受CPU影响无法饱和工作

内存计数器

计 数 器	计数器分析
Available Mbytes	可用物理内存数。如果Available Mbytes的值很小（100MB或更小），则说明计算机上总的内存可能不足，或某程序没有释放内存
Page/sec	表明由于硬件页面错误而从磁盘取出的页面数，或由于页面错误而从磁盘取出的页面数，或由于页面错误而写入磁盘以释放内存空间的页面数。一般如果Page/sec持续高于几百，那么应该进一步研究页交换活动。有可能需要增加内存，以减少换页的需求（可以把这个数字乘以4000就得到由此引起的硬盘数据流量）。Page/sec的值很大，不一定表明内存有问题，而可能是运行使用内存映射文件的程序所致
Page read/sec	指页的硬故障，是Page/sec的子集，为了解析对内存的引用，必须读取页文件的次数。该数值越低越好，大于5时表示磁盘读而不是缓存读
Page Faults/sec	指每秒钟软性页面失效的数目（包括有些可以直接在内存中满足而有些需要从硬盘中读取），而Page/sec只表明数据不能在指定内存中立即使用
Cache Bytes	文件系统缓存（File System Cache），默认情况下为50%的可用物理内存。如果怀疑有内存泄漏，请监视Memory\Available Bytes和Memory\Committed Bytes，以观察内存行为，并监视可能泄漏内存的进程Process\Private Bytes、Process\Working set和Process\Handle Count。如果怀疑是内核模式进程导致了泄露，则还应该监视Memory\Pool Nonpaged Bytes、Memory\Pool Nonpaged Allocs和Process(process_name)\Pool Nonpaged Bytes
Pages /sec	每秒钟检索的页数。该数字应少于每秒1页
Committed Bytes	以字节表示的确认虚拟内存。确认内存磁盘页面文件上保留了空间的物理内存。每个物理磁盘上可以有一个或一个以上的页面文件。这个计数器只显示上一回观察到的值；它不是一个平均值。其实就是指有多少虚拟内存正在被使用，虚拟内存是占用硬盘空间的内存，和物理内存无关

磁盘计数器

计 数 器	计数器分析
%Disk Time	指所选磁盘驱动器为读或写入请求提供服务所用时间的百分比。如果三个计数器都比较大，那么硬盘不是瓶颈。如果只有%Disk Time比较大，另外两个都比较适中，则硬盘可能会是瓶颈。若数值持续超过80%，则可能是内存泄漏。如果在Windows下出现无法读取磁盘性能信息的情况，则可以在命令行窗口中运行diskperf –yD命令，启动磁盘性能计数器
Avg.Disk Queue Length	它指的是当前磁盘的队列长度。通俗点来解释就是：计数器反映的磁盘完成请求所用的时间。较高的值表明磁盘控制器由于失败而不断重试该磁盘。这些故障会增加平均磁盘传送时间。 该值应不超过磁盘数的1.5倍，要提高性能，可增加磁盘。 注意：一个Raid Disk实际有多个磁盘
Average Disk Read/Write Queue Length	指读取（写入）请求（列队）的平均数
Disk Reads(Writes)/sec	物理磁盘上每秒磁盘读/写的次数。两者相加，应小于磁盘设备最大容量
Average Disk sec/Read	指以秒计算的在此盘上读取数据所需的平均时间
Average Disk sec/Transfer	指以秒计算的在此盘上写入数据所需的平均时间。 一般来说，定义该值小于15ms最佳，介于15～30ms为良好，30～60ms为可以接受，超过60ms则需要考虑更换硬盘或硬盘的RAID方式了
Bytes Total/sec	发送和接收字节的速率，包括帧字符在内。判断网络连接速度是否是瓶颈，可以用该计数器的值和目前网络的带宽进行比较

线程计数器

计 数 器	计数器分析
Context Switch/sec	如果决定要增加线程字节池的大小，应该同时监视实例化inetinfo和dllhost进程这两个计数器。增加线程可能会增加上下文切换次数，这样性能不会上升，反而会下降。如果多个实例的上、下文切换值非常高，就应该减小线程字节池

进程计数器

计 数 器	计数器分析
Process/ %Processor Time	被处理器消耗的处理器时间数量。如果服务器专用于SQL Server，则可接受的最大上限是80%～85%
Page Faults/sec	将进程产生的页故障与系统产生的相比较，以判断这个进程对系统页故障产生的影响
Working set	处理线程最近使用的内存页，反映了每个进程使用的内存页的数量。如果服务器有足够的空闲内存，页就会被留在内存中，当自由内存少于一个特定的阈值时，页就会被清除内存
Process/Private Bytes	指进程所分配的无法与其他进程共享的当前字节数量。该计数器主要用来判断进程在性能测试过程中有无内存泄漏。例如：对于一个IIS之上的Web应用，我们可以重点监控inetinfo进程的Private Bytes，如果在性能测试过程中，该进程Private Bytes计数器的值不断增加，或是性能测试停止后一段时间，该进程的Private Bytes仍然持续较高水平，则说明应用存在内存泄漏

SQLServer计数器

计 数 器	计数器分析
Access Methods:Full Scans/sec	每秒不受限的完全扫描数。可以是基本表扫描或全索引扫描。如果这个计数器显示的值比1或2高，应该分析查询以确定是否确实需要全表扫描，以及SQL查询是否可以被优化
Access Methods:Page splits/sec	由于数据更新操作引起的每秒页分割的数量
GeneralStatistics:Logins/sec	每秒登录到SQL Server的计数
General Statistics:User Connections	计数器显示用户连接SQL Server数。它的最高值是255。计数器的值增长的原因可能是性能问题和吞吐量的影响。数据库管理员应监测此计数器，以解决性能问题
Buffer Manager	用来监视SQL Server如何使用内存存储数据页、内部数据结构和过程高速缓存。在SQL Server从磁盘读取数据库页和将数据库页写入磁盘时监视物理I/O。监视SQL Server所使用的内存和计数器，有助于确定是否由于缺少可用物理内存存储高速缓存中经常访问的数据而导致瓶颈。如果是这样，SQL Server必须从磁盘检索数据，以及考虑是否可以通过添加更多内存，使更多内存可用于数据高速缓存或SQL Server内部结构来提高查询性能
Buffer Manager:Page Reads/sec	指每秒发出的物理数据库页读取数。这一统计信息显示的是在所有数据库间的物理页数读取总数。由于物理I/O的开销大，可以通过使用更大的数据高速缓存、智能索引、更高效的查询或者改变数据库设计等方法，使开销减到最小
Buffer Manager:Page Writes/sec	指每秒执行的物理数据库写的页数
Buffer Manager:Buffer Cache Hit Ratio	该比率是缓存命中总次数与缓存查找总次数之比。经过很长时间后，该比率的变化很小。由于从缓存中读取数据比从磁盘中读取数据的开销小得多，一般希望该比率高一些。通常可以通过增加SQL Server 的可用内存量来提高缓冲区高速缓存命中率。该值依应用程序而定，但比率最好为90%或更高。可增加内存直到这一数值持续高于90%，即表示90%以上的数据请求可以从数据缓冲区获得所需数据

SQLServer计数器

Buffer Manager: Lazy Writes/sec	惰性写进程每秒写的缓冲区的数量，其值最好为0
Cursor Manager by Type: Cache Hit Ratio	提供计数器（按类型分组）来监视游标：高速缓存命中次数和查找次数的比率。Cache可以包括Log Cache、Buffer Cache以及Procedure Cache。Cache Hit Ratio是一个总体的比率，是高速缓存命中次数和查找次数的比率。其对查看SQL Server高速缓存如何提升有效的系统性能是一个非常好的计数器。如果这个值持续低于80%，就需要增加更多的内存
SQL Statistics:Batch Requests/sec	指每秒收到的Transact-SQL命令批数。这一统计信息受所有约束（如I/O、用户数、高速缓存大小、请求的复杂程度等）影响，批请求数高意味着吞吐量很好
Cache Manager：	用于监视SQL Server如何使用内存存储对象，如存储过程、特殊和准备好的Transact-SQL语句及触发器
Cache Manager：Cache Hit Ratio（All Instances）	显示在高速缓存中找到数据的命中率。如果数值持续小于85%，则表示内存有问题
Latches	用于监视称为“闩锁”的内部SQL Server资源锁。监视闩锁以明确用户活动和资源使用情况，有助于查明性能瓶颈
Latches:Average Latch Wait Time(ms)	一个SQL Server线程必须等待一个闩的平均时间，以毫秒为单位。如果这个值很高，则系统可能正经历严重的竞争问题
Latches:Latch Waits/sec	在闩上每秒的等待数量。如果这个值很高，则表明系统正经历严重的竞争问题

SQLSever计数器

Locks	提供有关个别资源类型上的SQL Server锁的信息。锁加在SQL Server资源上，以防止多个事务并发使用资源。例如，有一个排他锁被一个事务加在某一表的某一行上，在这个锁被释放前，其他事务都不可以修改这一行。应尽可能少使用锁，可提高并发性，从而改善性能。可以同时监视Locks对象的多个实例，每个实例代表一个资源类型上的一个锁
Locks:Lock Waits/sec	显示在当前进程完成之前强制其他进程等待的每秒锁定请求的数量。如果该值始终大于0，则表示事务有问题
Locks:Number of Deadlocks/sec	导致死锁的锁请求数量
Locks:Average Wait Time(ms)	线程等待某种类型的锁的平均等待时间
Locks:Lock Requeste/sec	每秒某种类型的锁请求数量
Memory Manager	用于监视服务器内存总体的使用情况，以估计用户活动和资源使用，有助于查明性能瓶颈。监视SQL Server实例所使用的内存，有助于确定是否由于缺少可用物理内存存储高速缓存中经常访问的数据而导致瓶颈。如果这样，SQL Server必须从磁盘检索数据，以及考虑是否可以通过添加更多内存，或使更多内存可用于高速数据缓存或SQL Server内部结构，来提高查询性能
Memory Manager:Lock Blocks	指服务器上锁定块的数量，锁是加在页、行或者表这样的资源上。通常不希望看到此值增长
Memory Manager:Total Server Memory	指SQL Server服务器当前正在使用的动态内存总量
Disk I/O	指SQL Server从磁盘读取数据的频率。与其他操作相比，例如内存访问、物理I/O会耗费大量时间。尽可能减少物理I/O可以提高查询性能

缓存计数器

计 数 器	计数器分析
File Cache Hits	文件缓存命中的具体值
File Cache Hits %	全部缓存请求中缓存命中次数所占的比例，反映了IIS 的文件缓存设置的工作情况。对于一个大部分是静态网页组成的网站，该值应该保持在80%左右
File Cache Flushes	自服务器启动之后文件缓存刷新的次数，如果刷新太慢，会浪费内存；如果刷新太快，缓存中的对象会被频繁地丢弃再重新生成，起不到缓存的作用。通过将File Cache Hits除以File Cache Flushes可以得出缓存命中率对缓存清空率的比率。通过观察这两个值，可以得到一个适当的刷新值（设置IIS中的ObjectTTL 、MemCacheSize、MaxCacheFileSize计数值）
Connection Refused	该数值越低越好，高数值表明网络适配器或处理器存在瓶颈

IIS计数器

计 数 器	计数器分析
POST Request/Sec	使用POST方法的HTTP请求速率，POST请求用于表格或网关请求
Not Found Errors	显示由于被请求文件无法找到而导致的服务器无法回应的请求数（HTTP状态代码404）
Bytes Sent/Sec	“送出字节数/秒”是Web服务送出数据字节的比率
Bytes Received/Sec	“接收字节数/秒”是Web服务接收数据字节的比率
GET Request/Sec	使用GET方法的HTTP请求速率。GET请求用于基本文件获取或图像地图，它们可以和表格一起使用
Maximum Connections、Total Connection Attempts	Maximum Connections：“最大连接数”是和Web服务同时建立起的最大连接数。Total Connection Attempts：“连接尝试总数”是从服务启动时利用Web服务尝试连接的总数。该计数器应用于所列的全部实例

网络计数器

计 数 器	计数器分析
Bytes Total/sec	发送和接收字节的速度，包括帧字符在内。判断网络连接速度是否为瓶颈，可以用该计数器的值和目前网络的带宽相除，结果应该小于50%

ASP.Net计数器

计 数 器	计数器分析
Request/Sec 、Request Executing	每秒执行的请求数和当前执行的请求数。如果Request/Sec 的值比较小，Web程序可能是瓶颈
Request Wait Time、Request Executing Time 、Request Queued	最近的请求在队列中等待的毫秒数、执行最近的请求所用的毫秒数、等待处理的请求数。该计数器应保持接近0。若超过IIS队列长度，则会出现“服务器忙”错误。Request Wait Time和Request Queued在该项状况下应该接近0，如果这两个值太大，那么需要重写代码以提高性能

Apache计数器

计 数 器	计数器分析
Apache CPU Usage apache	服务器CPU的占用率
Kbytes Sent/sec	服务器每秒发送的字节数
Hits/sec	Apache服务每秒的点击率
#Busy Workers	Apache服务占用率
#Idle Workers	Apache服务空闲率

MYSQL计数器

计 数 器	计数器分析
Threads_connected	表示当前有多少个客户连接该MySQL服务器，连接数是否过多，网络是否存在问题，它是动态变化的。当threads_connected == max_connections 时，数据库系统就不能提供更多的连接数了，这时，如果程序还想新建连接线程，数据库系统就会拒绝，如果程序没做太多的错误处理，就会出现报错信息
Threads_running	如果数据库超负荷了，将会得到一个正在（查询的语句持续）增长的数值。这个值也可以小于预先设定的值。这个值在很短的时间内超过限定值是没问题的若超过预设值时且5秒内没有回落，就要同时监视其他的一些值
Aborted_clients	客户端被异常中断的数值（因为连接到MySQL服务器的客户端没有正常地断开或关闭）。对于一些应用程序是没有影响的，但对于另一些应用程序可能要跟踪该值，因为异常中断连接可能表明一些应用程序有问题
Questions	每秒获得的查询数量。也可以是全部查询的数量（注：可以根据输入不同的命令得到你想要的不同的值）
Handler_*	如果想监视底层（low-level）数据库负载，这些值是值得去跟踪的。如果Handler_read_rnd_next值与正常值相差悬殊，可能是优化或索引出问题了。Handler_rollback表明事务被回滚的查询数量
Opened_tables	指表缓存没有命中的数量。如果该值很大，就需要增加table_cache的数值

MYSQL计数器

Select_full_join	没有主键（key）联合（Join）的执行。该值可能是零。这是捕获开发错误的好方法，因为这样的查询有可能降低系统的性能
Select_scan	执行全表搜索查询的数量（注：Select_scan除以总查询数量的商应该是常数）。如果发现该值持续增长，说明需要优化，缺乏必要的索引或其他问题
Slow_queries	超过该值（--long-query-time）的查询数量，或没有使用索引查询数量。对于全部查询会有小的冲突。如果该值增长，则表明系统有性能问题
Threads_created	该值一般较低。较高的值可能意味着需要增加thread_cache的数值，或遇到了持续增加的连接，表明存在潜在的问题
Pending normal aio reads	该值是innodb io请求查询的大小（size）。如果该值超出了10～20的范围，可能存在一些瓶颈

Oracle计数器

计 数 器	计数器分析
Buffer Nowait %	缓冲区未等待率，指在缓冲区中获取Buffer的未等待比率。该指标的值应接近100%，如果该值较低，则可能要增大Buffer Cache
Redo Nowait %	Redo缓冲区未等待率，指在Redo缓冲区获取Buffer的未等待比率。该指标的值应接近100%，如果该值较低，则有以下两种可能的情况：
	online redo log没有足够的空间
	log切换速度较慢
Buffer Hit %	缓冲区命中率，该指标的值通常应在90%以上，否则需要调整。如果持续小于90%，可能要加大db_cache_size。但有时，缓存命中率低并不意味着Cache设置小了，可能是潜在的全表扫描降低了缓存命中率
In-memory Sort %	内存排序率，排序操作在内存中进行的比率。当查询需要排序的时候，数据库会话首先选择在内存中进行排序，当内存大小不足时，将使用临时表空间进行磁盘排序，但磁盘排序效率和内存排序效率相差好几个数量级。该指标的值应接近100%，如果指标的值较低，则表示出现了大量排序时的磁盘I/O操作，可考虑加大sort_area_size参数的值

Oracle计数器

Library Hit %	共享区命中率，该指标主要代表SQL在共享区的命中率。其值通常应在95%以上，否则需要考虑加大共享池（修改shared_pool_size参数值），绑定变量，修改cursor_sharing等参数
Soft Parse %	软解析的百分比，该指标是指在Oracle对SQL的解析过程中，软解析所占的百分比。软解析（Soft Parse）是指当Oracle接到Client提交的SQL后会首先在共享池（Shared Pool）里面去查找是否有之前已经解析好的与刚接到的这个SQL完全相同的SQL。当发现有相同的SQL就直接用之前解析好的结果，这就节约了解析时间以及解析时候消耗的CPU资源。该指标的值通常应在95%以上，如果低于80%，那么可能SQL基本没被重用，SQL没有绑定变量，需要考虑绑定变量
Latch Hit %	闩命中率，获得Latch的次数与请求Latch的次数的比率。该指标的值应接近100%，如果低于99%，可以考虑采取一定的方法来降低对Latch的争用
Execute to Parse %	SQL语句执行与解析的比率，SQL语句执行与解析的比率。SQL语句一次解析后执行的次数越多，该比率越高，说明SQL语句的重用性很好。该指标的值应尽可能高，如果过低，则可以考虑设置session_cached_cursors参数
Memory Usage %	共享池内存使用率，该指标是指在采集点时刻，共享池（Share Pool）内存被使用的比例。该指标的值应保持在75%～90%，如果这个值太低，就会造成内存浪费；如果太高，则会使共享池外部的组件老化。如果SQL语句被再次执行，则会发生硬分析
db file scattered read (cs)	文件分散读取，该等待事件通常与全表扫描有关。因为全表扫描是被放入内存中进行的，通常情况下它不可能被放入连续的缓冲区中，所以散布在缓冲区的缓存中。如果这个等待事件比较显著，则可能说明对于某些全表扫描的表，没有创建合适的索引。尽管在特定条件下执行全表扫描可能比索引扫描更有效，但如果出现这种等待时，最好检查这些全表扫描是否有必要
db file sequential read (cs)	文件顺序读取，该等待事件通常与单个数据块相关的读取操作有关。如果这个等待事件比较显著，则可能表示在多表连接中，表的连接顺序存在问题，或者可能不合适地使用了索引。对于调整良好的系统，这一数值大多是很正常的，但在某些情况下，它可能暗示着系统中存在问题。应检查索引扫描，以保证每个扫描都是必要的并检查多表连接的顺序

Oracle计数器

buffer busy (cs)	缓冲区忙，当一个会话想要访问缓存中的某个块，而这个块正在被其他会话使用时，将会产生该等待事件。这时候，其他会话可能正在从数据文件向缓存中的这个块写入信息，或正在对这个块进行修改。出现这个等待事件的频度不应大于1%。如果这个等待事件比较显著，则需要根据等待事件发生在缓存中的位置（如字段头部、回退段头部块、回退段非头部块、数据块、索引块等）采取相应的优化方法
enqueue (cs)	enqueue是一种保护共享资源的锁定机制。该锁定机制保护共享资源（如记录中的数据）以避免两个人在同一时间更新同一数据。enqueue包括一个排队机制，即FIFO（先进先出）排队机制。注意：Oracle的Latch机制不是FIFO。Enqueue等待通常指的是ST enqueue、HW enqueue、TX4 enqueue和TM enqueue。如果enqueue等待事件比较显著，则需要根据enqueue等待类型，采取相应的优化方法
闩释放 (latch free (cs))	该等待事件意味着进程正在等待其他进程已持有的Latch。
	Latch是一种低级排队机制（准确地称为相互排斥机制），用于保护系统全局区域（SGA）中共享内存结构。Latch就像是一种快速地被获取和释放的内存锁。Latch 用于防止共享内存结构被多个用户同时访问。对于常见的Latch等待通常的解决方法包括
	Share Pool Latch：在OLTP应用中应该更多地使用绑定变量以减少该Latch的等待。
	Library Cache Latch：需要通过优化SQL语句使用绑定变量减少该Latch的等待

WebLogic计数器

计 数 器	计数器分析
MessagesLogged	该WebLogic服务器实例产生的日志消息总数
Registered	禁用代理中的缓存的专用属性
CachingDisabled	返回本服务器上登记的失效的总数
ServerRuntime计数器	
SocketsOpenedTotalCount	返回本服务器上登记的socket的总数
OpenSocketsCurrentCount	返回本服务器上当前登记的socket数
ActivationTime	返回服务器激活时间
AdminServerListenPort	返回管理服务器监听的端口
RestartsTotalCount	返回自群集上次激活后本服务器重启的总次数
ListenPort	返回当前服务器监听连接的端口
ExecutionTimeAverage	返回所有servlet自被创建后被调用的平均数
ExecutionTimeHigh	返回servlet自创建后耗时最长的一个交易调用的时间

WebLogic计数器

ReloadTotalCount	返回servlet被重载总次数
PoolMaxCapacity	返回单线程模式下servlet的最大能力
InvocationTotalCount	返回servlet被调用的总次数
ExecutionTimeLow	返回servlet自创建后单次最短调用消耗的时间
ExecutionTimeTotal	返回servlet自创建后所有被调用的总时间数
WebAppComponentRuntime计数器	
SessionsOpenedTotalCount	返回该服务器上打开的会话总数
OpenSessionsHighCount	返回本服务器上打开会话总数的最大值
OpenSessionsCurrentCount	返回当前component打开的会话总数
EJBStatefulHomeRuntime计数器	
Stateless	如果mbean代表无状态会话bean，返回true，否则返回false
TransactionsCommittedTotalCount	返回bean的Home接口提交的交易总数（EJBStatelessHomeRuntime、EJBComponentRuntime、EJBStatefulHomeRuntime）
TransactionsInFlightTotalCount	返回bean的Home接口正在运行的交易总数(EJBStatelessHomeRuntime、EJBComponentRuntime、EJBStatefulHome- Runtime)
CachedBeansCurrentCount	返回当前缓存中bean的数目
TransactionsRolledBackTotalCount	返回bean的Home接口回滚的交易总数

WebLogic计数器

JTARuntime计数器	
TransactionsRolledBackTotalCount	返回回滚的交易数
SecondsActiveTotalCount	返回所有提交交易的总秒数
TransactionRolledBackAppTotalCount	返回因应用错误回滚的交易数
TransactionTotalCount	返回所有处理的交易总数，包含回滚、提交和引发其他交易的交易
TransactionRolledBackTimeoutTotalCount	返回因超时回滚的交易数
TransactionRolledBackSystemTotalCount	返回因系统内部错误回滚的交易数
TransactionHeuristicsTotalCount	返回完成的heuristic状态的交易数

WebLogic计数器

TransactionCommittedTotalCount	返回提交的交易数
TransactionRolledBackResourceTotalCount	返回因资源错误回滚的交易数
JVMRuntime计数器	
HeapSizeCurrent	返回当前JVM堆中内存数，单位是字节
HeapFreeCurrent	返回当前JVM堆中空闲内存数，单位是字节
EJBEntityHomeRuntime计数器	
TransactionsCommittedTotalCount	返回bean的Home接口提交的交易总数
TransactionsInFlightTotalCount	返回bean的Home接口运行中的交易总数
CachedBeansCurrentCount	返回当前缓存中的bean数目
TransactionsRolledBackTotalCount	返回bean的Home接口回滚的交易总数
EJBComponentRuntime计数器	

WebLogic计数器

Status	返回部署状态。不会使用EJBDeployment 接口中定义的状态集（DEPLOYED、UNDEPLOYED、ERROR），也不必要
JDBCConnectionPoolRuntime计数器	
WaitingForConnectionHighCount	返回本JDBCConnectionPoolRuntimeMBean 上最大等待连接数
WaitingForConnectionCurrentCount	返回当前等待连接的总数
ConnectionsTotalCount	返回自连接池实例化后的JDBC连接的总数
MaxCapacity	返回JDBC池的最大能力
WaitSecondsHighCount	返回等待连接中的最长时间
ActiveConnectionsCurrentCount	返回当前活动连接总数
ActiveConnectionsHighCount	返回本次JDBCConnectionPoolRuntimeMBean 上最大活动连接数
ExecuteQueueRuntime计数器	

WebLogic计数器

ExecuteThreadCurrentIdleCount	返回队列中当前空闲线程数
PendingRequestOldestTime	返回队列中最长的等待时间
ServicedRequestTotalCount	返回被本队列处理的请求总数
PendingRequestCurrentCount	返回队列中等待的请求数
FragmentsSentCount	返回本服务器上向集群发出的总广播片断数
PrimaryCount	返回本地服务器上的基本对象数
FragmentsReceivedCount	返回本服务器上收到集群发出的总广播片断数
ResendRequestsCount	返回因集群中的服务器错过而重发的delta状态消息的数目
MulticastMessagesLostCount	返回向本服务器发送的且丢失的广播消息总数
AliveServerCount	返回当前集群中活动的服务器数目
ForeignFragmentsDroppedCount	返回源于其他群集而使用相同广播地址的片断数目

WebLogic计数器

JMSRuntime计数器	
ConnectionsTotalCount	返回本JMS服务器自上次重置后的总连接数
JMSServersCurrentCount	返回当前JMS服务的连接数
JMSServersTotalCount	返回自服务器启动后JMS服务的连接次数
ConnectionsCurrentCount	返回本JMS服务器上当前的连接数
JMSServersHighCount	返回自服务器启动后JMS服务的最大连接数
ConnectionsHighCount	返回本JMS服务器自上次重置后的最大连接数

WebLogic计数器

TimeServiceRuntime计数器	
ScheduledTriggerCount	返回当前活动的计划触发器数目
ExecutionCount	返回触发器执行的总数
ExecutionsPerMinute	返回每分钟触发器执行的平均数
ExceptionCount	返回执行计划触发器掷出异常的总数
EJBStatelessHomeRuntime计数器	
TimeoutTotalCount	返回客户端从空闲池中获得一个无状态会话的实例而等待超时的总次数
TransactionsRolledBackTotalCount	返回bean的Home接口回滚交易总数
TransactionsCommittedTotalCount	返回bean的Home接口提交交易总数
CachedBeansInUseCurrentCount	返回当前使用中的bean数目

WebLogic计数器

CachedBeansCurrentCount	返回当前缓存中的bean数目
TransactionsInFlightTotalCount	返回bean的Home接口正在运行中的交易总数
Stateless	如果mbean代表一个无状态bean，则返回true，否则返回false
CachedBeansIdleCurrentCount	返回定位了但是空闲的bean数目
WaiterTotalCount	返回客户端从空闲池中获得一个无状态会话的实例需要等待的总次数
ConnectionPoolCount	返回WLEL连接池配置的数目
ServerSecurityRuntime计数器	
UnlockedUsersTotalCount	返回在服务器上取消锁定的次数
InvalidLoginUsersHighCount	返回具有显著的无效服务器登录尝试的用户的最大数目
LoginAttemptsWhileLockedTotalCount	返回锁定用户时尝试对服务器进行无效登录的累计次数
LockedUsersCurrentCount	返回服务器上当前锁定的用户数
InvalidLoginAttemptsTotalCount	返回对服务器进行无效登录尝试的累计次数
UserLockoutTotalCount	返回在服务器上进行用户锁定的累计次数

Step List

1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*

分析&定位

- 人是高等生物，在于人有意识，认识世界，改造世界的能力。（参见人教版高二政治书上）
  - 对已知事务进行了解
  - 使用已知知识对事务进行揣摩
  - 给出自己的理解

- 分析的前提是了解，一个清晰的分析能够大大简化调优
- 当有了好的分析，调优方案自然就慢慢凸显

好的分析

- 正确的分析模型
- 正确的数据获取来源
- 正确的数据
- 正确的分析过程
- 清晰明了的结论

分析案例1

- 政府财政收入是多少？
- 主要的政府财政支出在哪里？
- 12000亿地方政府的三公支出是什么概念？
- 哪个最有可能贪污，哪个部门最清廉？

分析案例2

- 如何到达培训点
  - 地铁？
  - 公交？
  - 11路（走路）
  - Other？

效率 VS 成本

- 效率
  - 地铁
  - 出租车
- 成本
  - 走路
  - 自行车
- 完美的性价比？

好比自动化，其实所有的质量都是在考虑ROI（Return On Investment投资回报率）

性能分析首先要做的就是隔离

前端
在客户端上或者是网络上的开销我们称之为前端

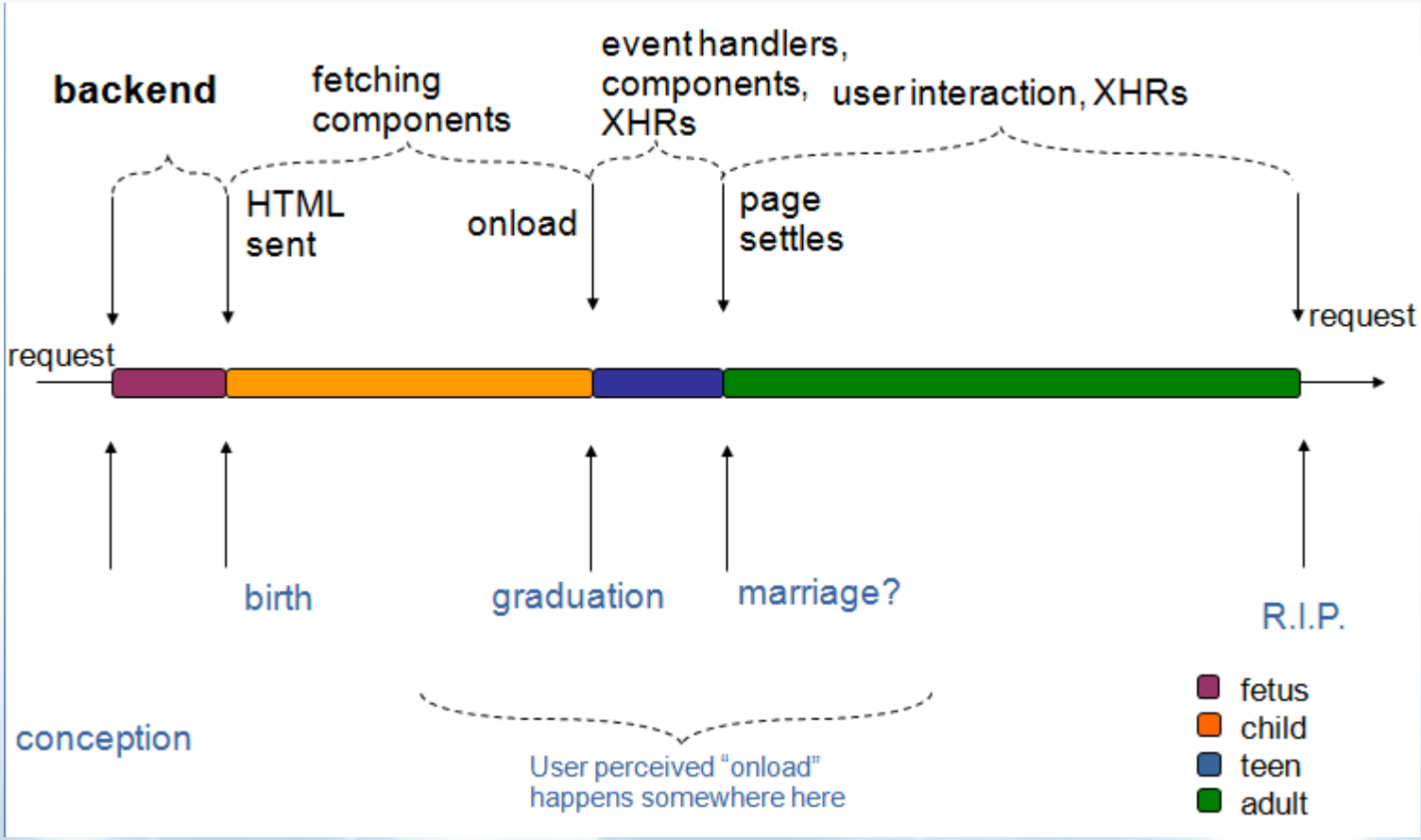
后端
在服务器端上的开销我们称之为后端

随着富客户端的技术普及，越来越多的问题出现在了前端。

富客户端技术为何流行？
1.更好的客户端体验
2.减少了服务器端的负载

问题？
1.浏览器兼容
2.开发难度
3.客户端配置要求
4.页面首次加载较大

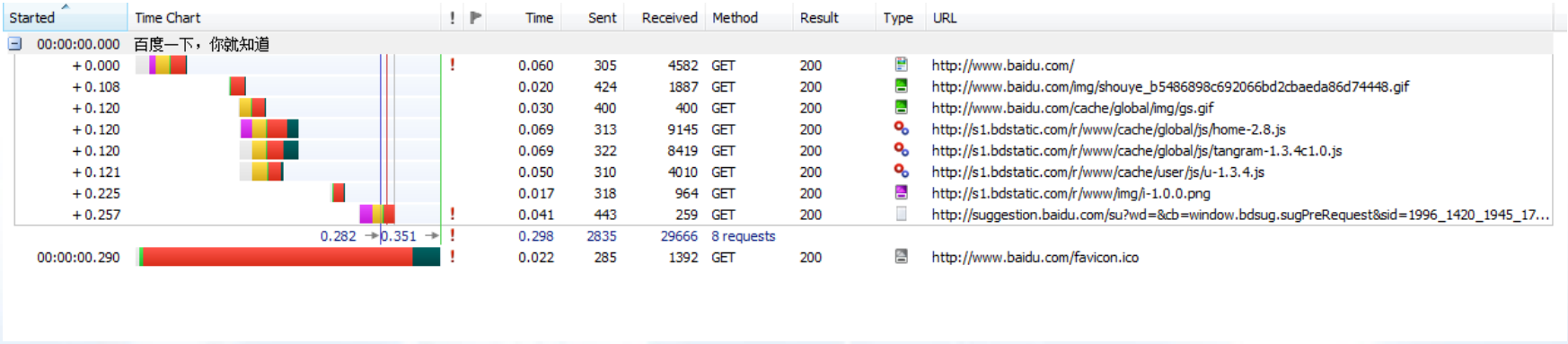
根据统计在整个网络应用中，响应时间在网络上的各种开销占到了整个响应时间的70%以上



前端跟踪1

如何进行初级前端跟踪

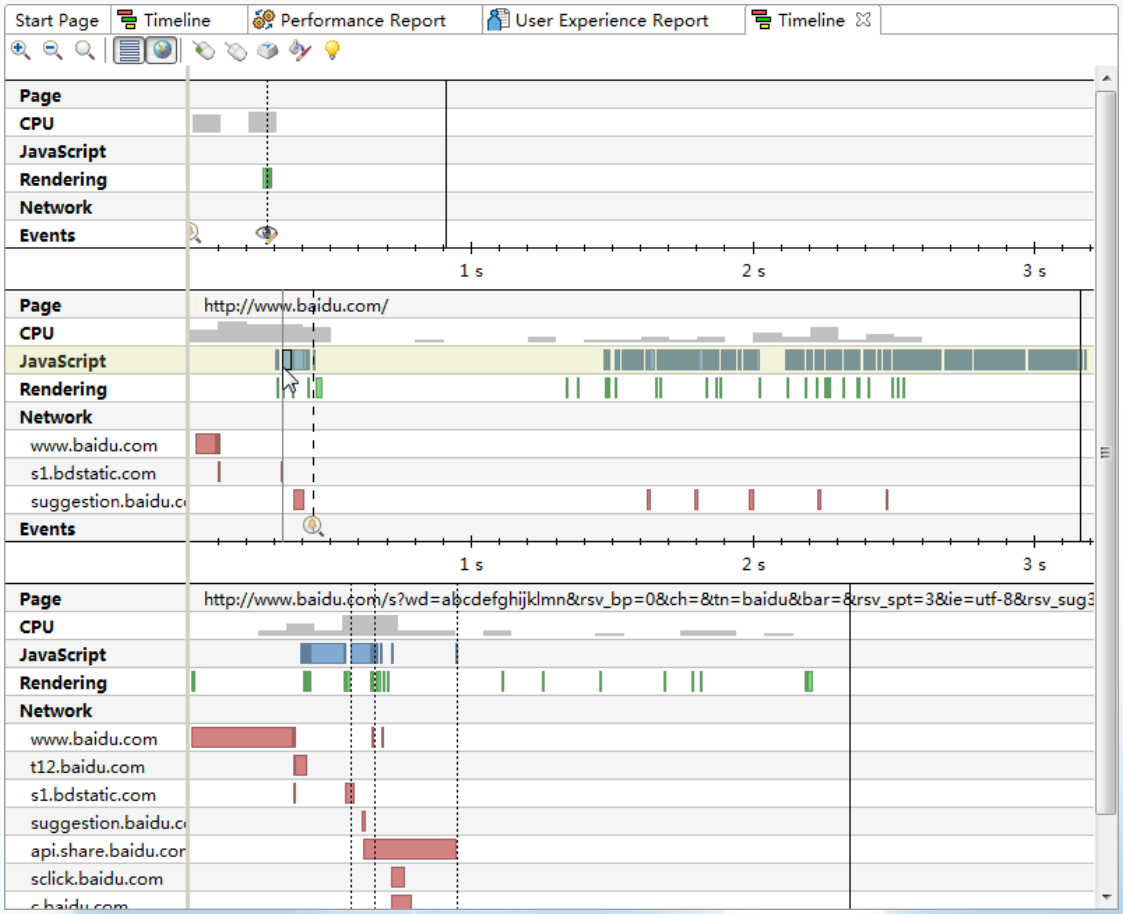
- **HTTPWATCH**
- **FireBug**
- **F12**



Started	Time Chart	!		Time	Sent	Received	Method	Result	Type	URL
00:00:00.000	百度一下，你就知道									
+ 0.000		!		0.060	305	4582	GET	200		http://www.baidu.com/
+ 0.108				0.020	424	1887	GET	200		http://www.baidu.com/img/shouye_b5486898c692066bd2cbaeda86d74448.gif
+ 0.120				0.030	400	400	GET	200		http://www.baidu.com/cache/global/img/gs.gif
+ 0.120				0.069	313	9145	GET	200		http://s1.bdstatic.com/r/www/cache/global/js/home-2.8.js
+ 0.120				0.069	322	8419	GET	200		http://s1.bdstatic.com/r/www/cache/global/js/tangram-1.3.4c1.0.js
+ 0.121				0.050	310	4010	GET	200		http://s1.bdstatic.com/r/www/cache/user/js/u-1.3.4.js
+ 0.225				0.017	318	964	GET	200		http://s1.bdstatic.com/r/www/img/i-1.0.0.png
+ 0.257		!		0.041	443	259	GET	200		http://suggestion.baidu.com/su?wd=&cb=window.bdsug.sugPreRequest&sid=1996_1420_1945_17...
	0.282 → 0.351 →	!		0.298	2835	29666	8 requests			
00:00:00.290		!		0.022	285	1392	GET	200		http://www.baidu.com/favicon.ico

如何进行高级前端（JavaScript）跟踪

- **DynaTrace**
- **PageSpeed**



前端调优

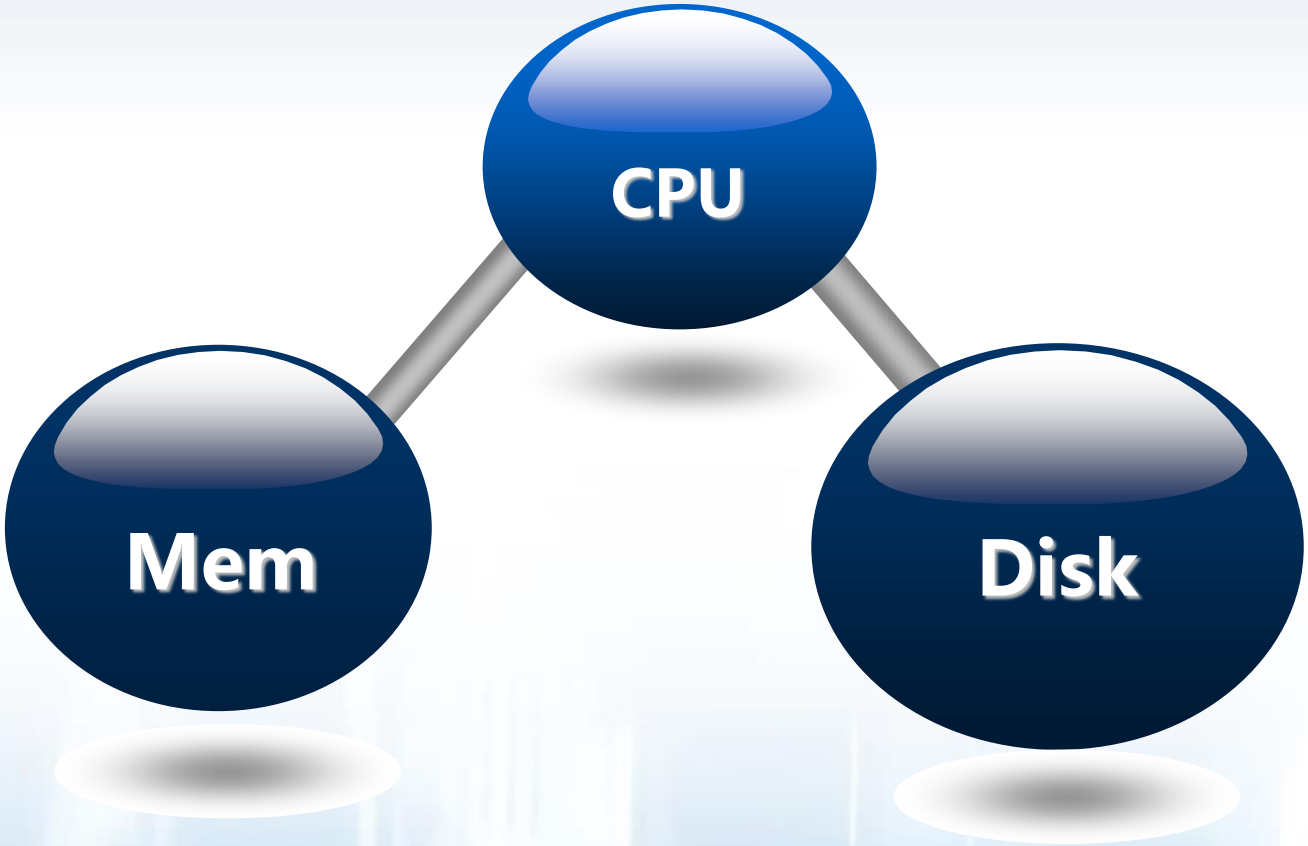
- 减少HTTP请求数据量及个数
- 使用CDN数据加速
- 合理使用Expires增加缓存效果
- 使用动态或静态压缩技术
- 在顶部加载CSS
- JS存放页面底部
- 使AJAX请求可缓存
- 使用外部JAVASCRIPT或CSS
- 减少DNS查询

- *http://internetsupervision.com

后端问题的问题往往更加简单，因为一切尽在手中。

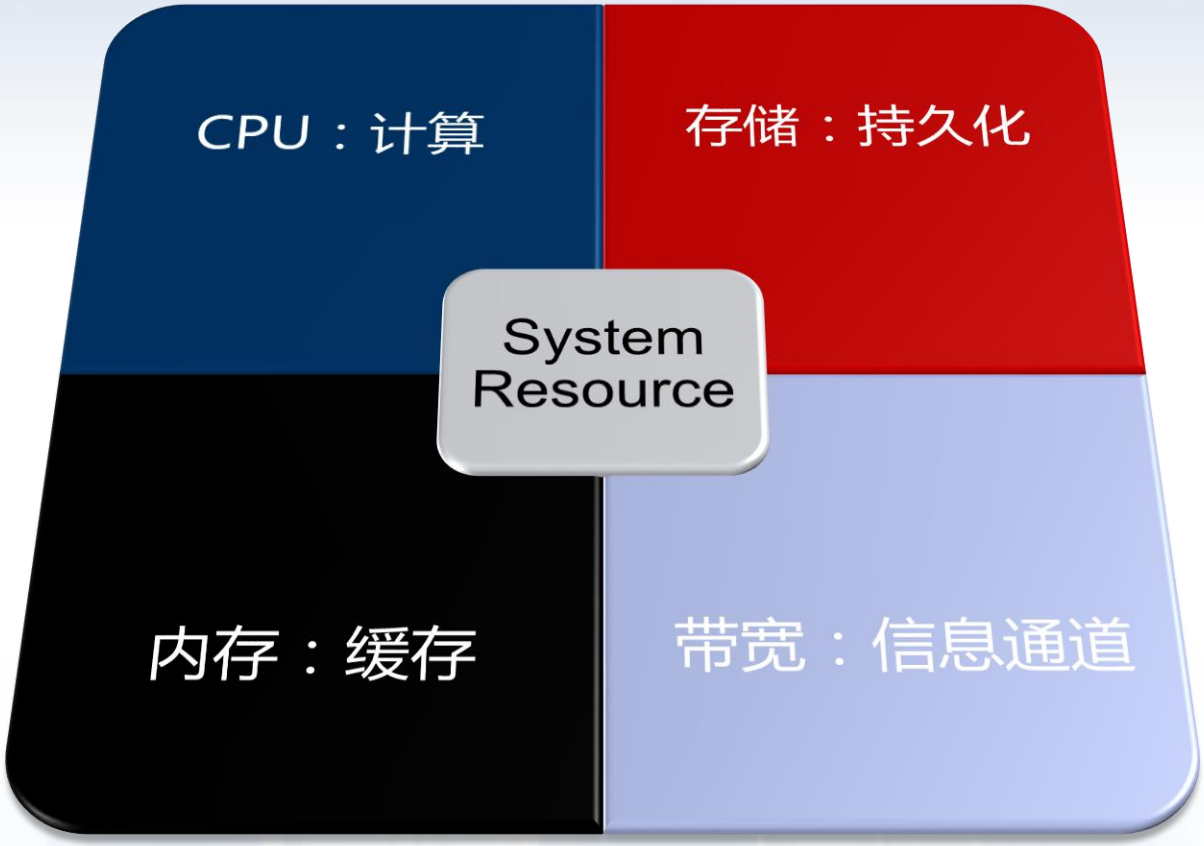
相对前端问题，后端问题看似复杂，但其实却更加简单，因为通过简单的负载就能重现出后端的性能问题，定位和调整自然有所方向。

系统性能瓶颈



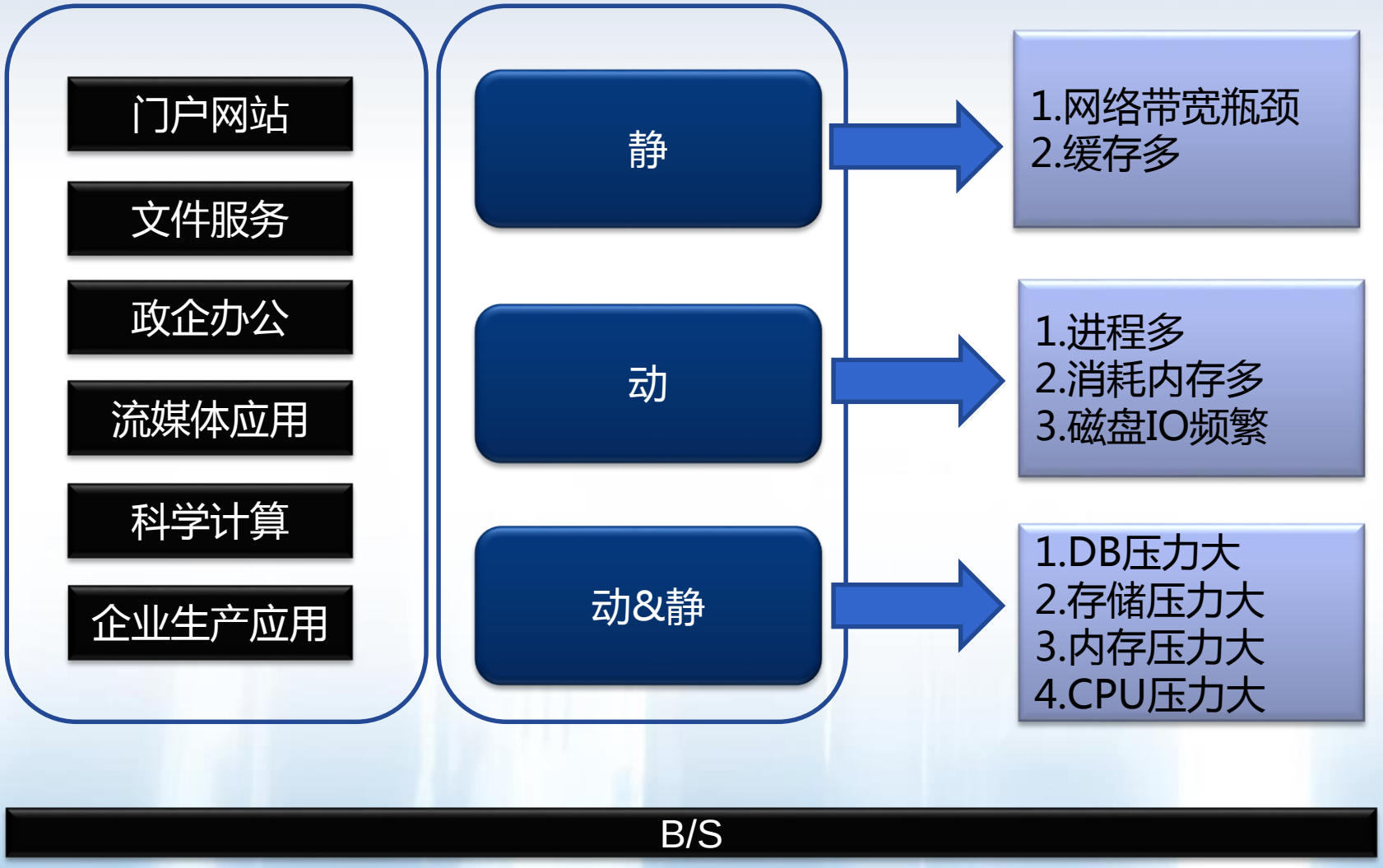
CPU进程等待 → 内存增加 → 虚拟内存使用 → 磁盘IO增加
CPU开销增加
(进程切换、缺页处理)

系统性能瓶颈

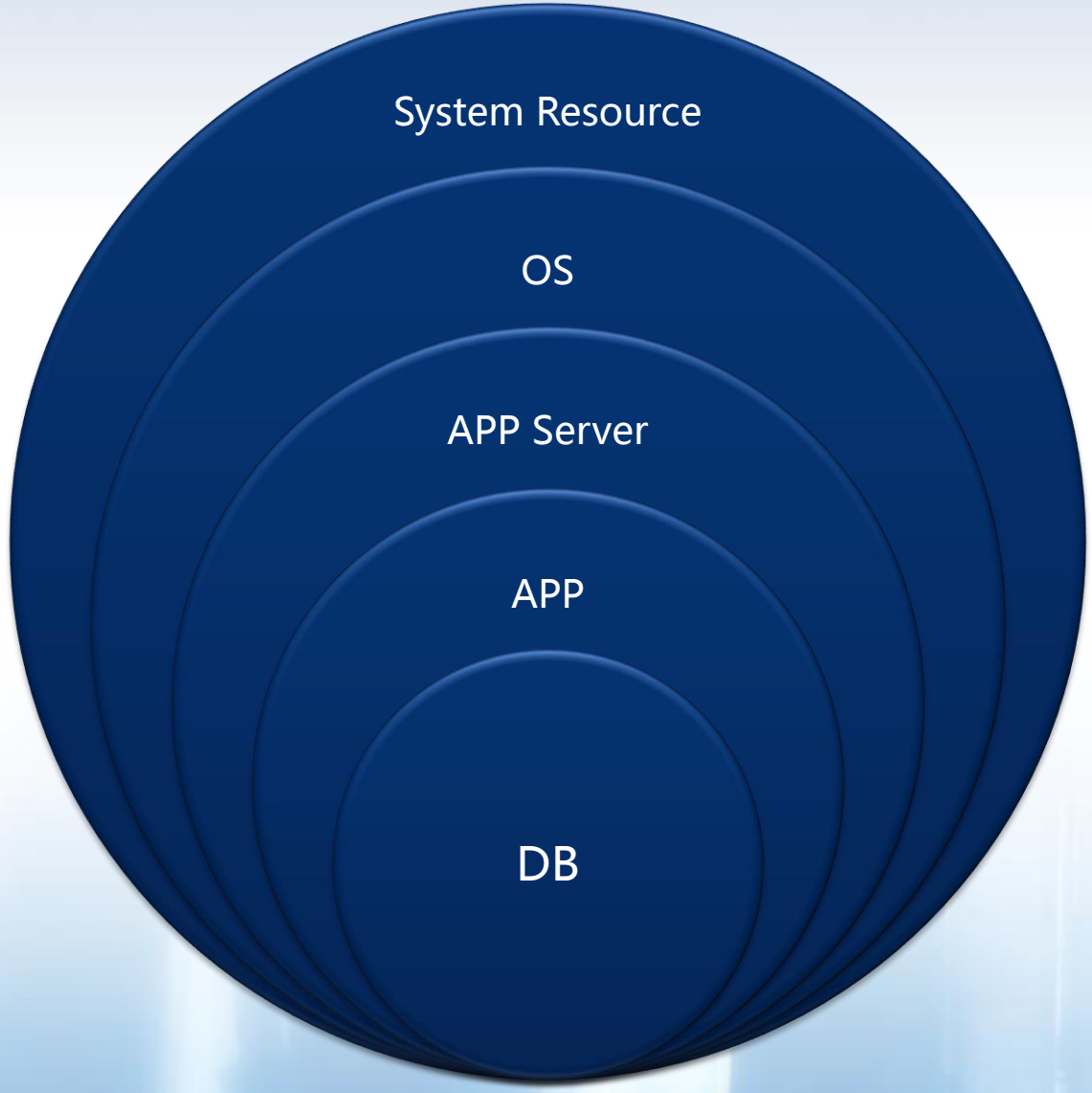


1.通过缓存来提高读写效率
2.调整存储
3.条带化

系统特征



性能表现



系统资源:
服务器、客户机CPU，内存，硬盘等配置

操作系统:
操作系统资源分配

应用服务:
中间件、中间件配置

应用程序:
程序性能

数据库:
参数配置

硬件级别的分析

- 常见硬件瓶颈（CPU，内存，磁盘，网络）分析
- 关键字：磁盘4K随即读写

中央处理器(CPU)		CPU 核...	主板	芯片組
16821 MB/秒	Core i7-3960X Extreme	3300 MHz	Intel DX79SI	X79
15984 MB/秒	Core i7-2600	3400 MHz	Asus P8P67	P67
14679 MB/秒	Core i7-990X Extreme	3466 MHz	Intel DX58SO2	X58
14327 MB/秒	FX-6100	3300 MHz	Asus Sabertooth 990FX	AMD990FX
13914 MB/秒	Core i7-965 Extreme	3200 MHz	Asus P6T Deluxe	X58
12464 MB/秒	Xeon X5550	2666 MHz	Supermicro X8DTN+	i5520
11593 MB/秒	Xeon X3430	2400 MHz	Supermicro X8SIL-F	i3420
9531 MB/秒	Core i5-650	3200 MHz	Supermicro C7SIM-Q	Q57 Int.
9054 MB/秒	Sempron 140	2700 MHz	Asus Sabertooth 990FX	AMD990FX
8831 MB/秒	Athlon64 X2 Black 6400+	3200 MHz	MSI K9N SLI Platinum	nForce570SLI
8042 MB/秒	Pentium EE 955	3466 MHz	Intel D955XBK	i955X
7964 MB/秒	A8-3850	2900 MHz	Gigabyte GA-A75M-UD2H	A75 Int.
7927 MB/秒	P4EE	3733 MHz	Intel SE7230NH1LX	iE7230
7914 MB/秒	Phenom II X6 1055T	2800 MHz	Gigabyte GA-790FXTA-UD5	AMD790FX
7622 MB/秒	Phenom II X4 Black 940	3000 MHz	Asus M3N78-EM	GeForce8300 Int.
7202 MB/秒	Core 2 Extreme QX9650	3000 MHz	Gigabyte GA-EP35C-DS3R	P35

OS级别的分析

- 常见系统瓶颈（硬件支持，连接线程，TCP连接池）
- 关键字：资源管理&监控

表示层级别的分析

- 常见表示层瓶颈（Apache，IIS，Nginx）
- 关键字：并发吞吐量，最大连接数

应用层级别的分析

- 常见应用层瓶颈（C#，Java，PHP）
- 应用服务器配置
- 关键字：代码复杂度

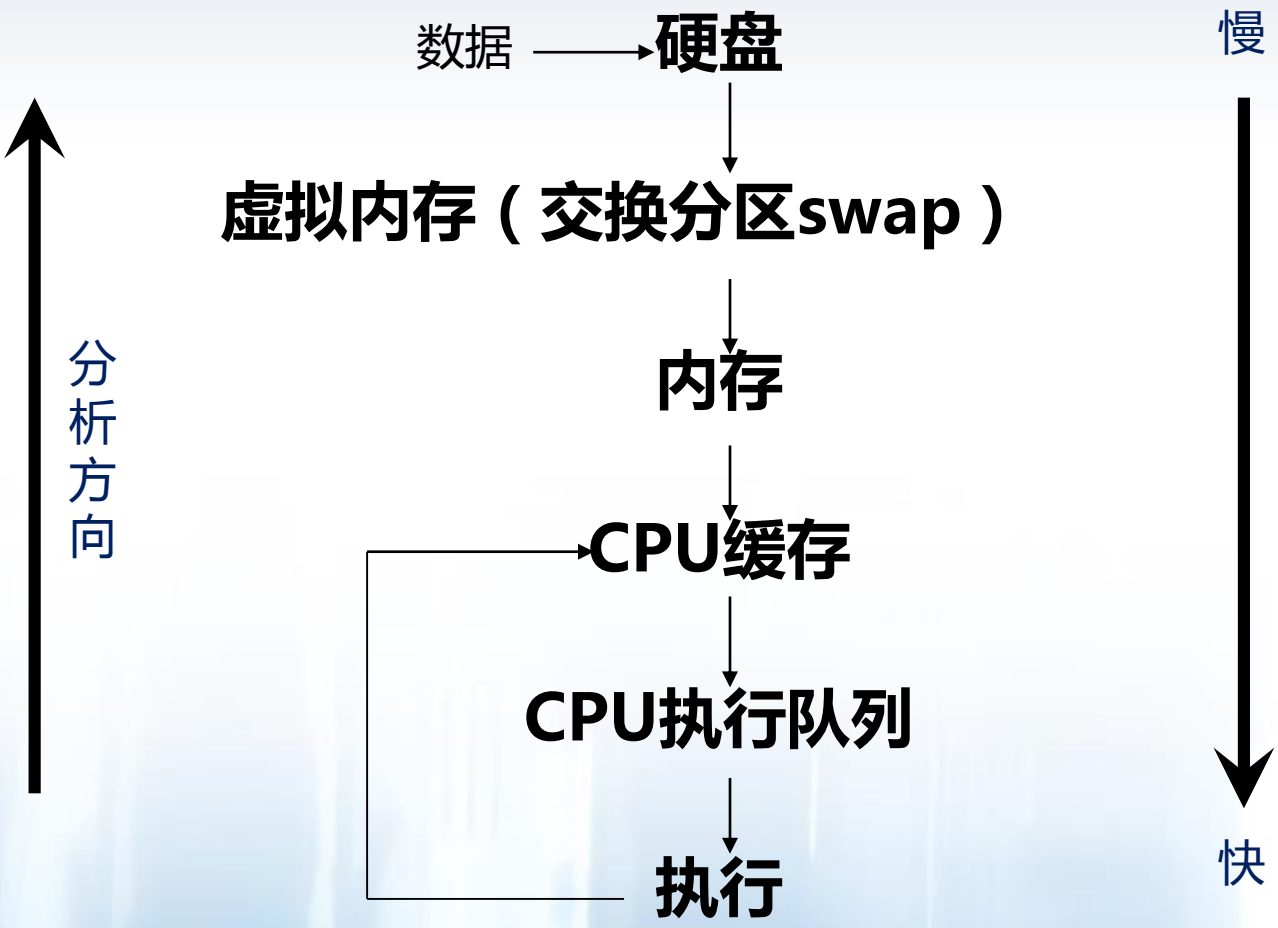
数据层级别的分析

- 常见数据层瓶颈（Oracle，SQLServer，Mysql）
- 关键字命中率、执行计划

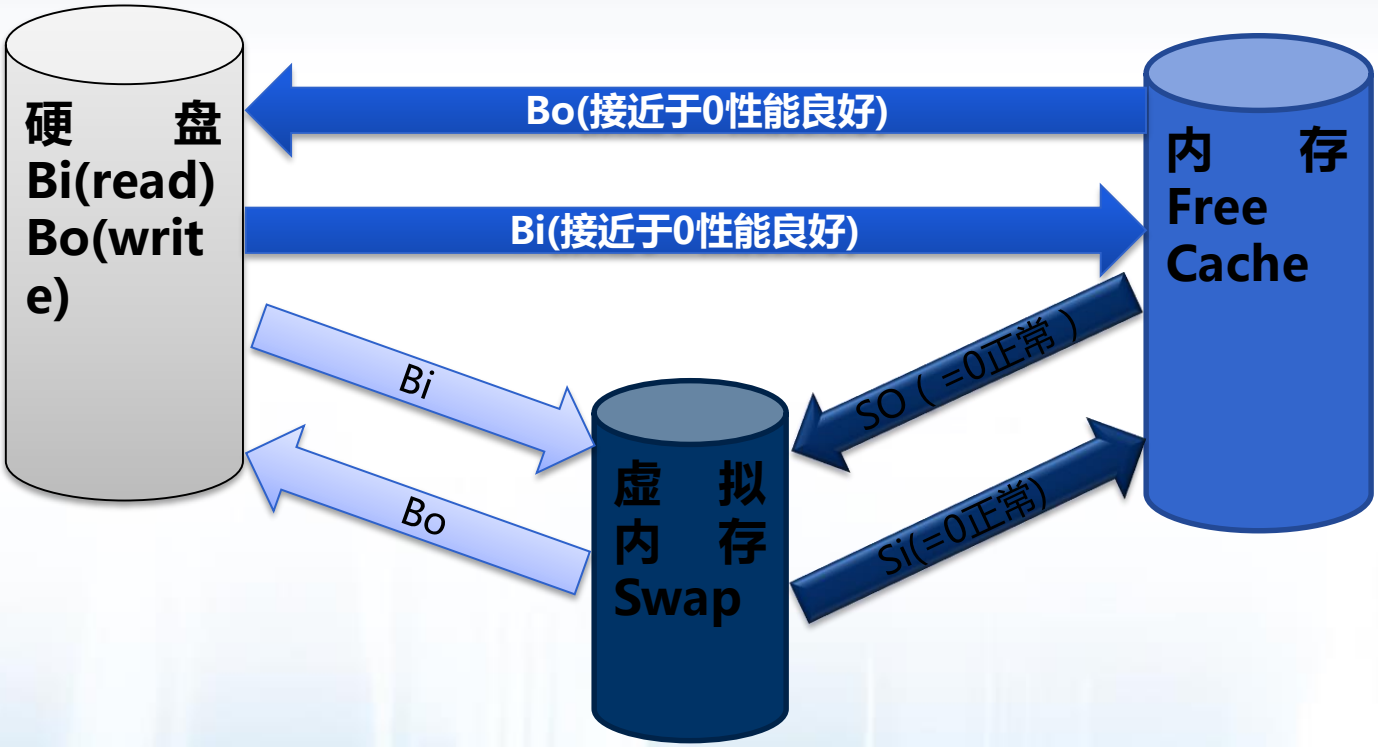
性能瓶颈

- 应用程序及服务器瓶颈
  - 内存占用过多、内存泄露：声明的对象多、程序算法欠佳
  - CPU占用率过大 ：任务多、运算量大
  - 优化算法、优化业务 过程、硬件升级
- 数据库及服务器瓶颈（Oracle为例）
  - App 请求数及请求涉及的数据返回 量大
  - DB中SGA、PGA、高速Cache设置（ASM 自动存储管理）
  - DB 磁盘操作 效率低下 （摩尔定律失效，读写磁盘是机械运动）

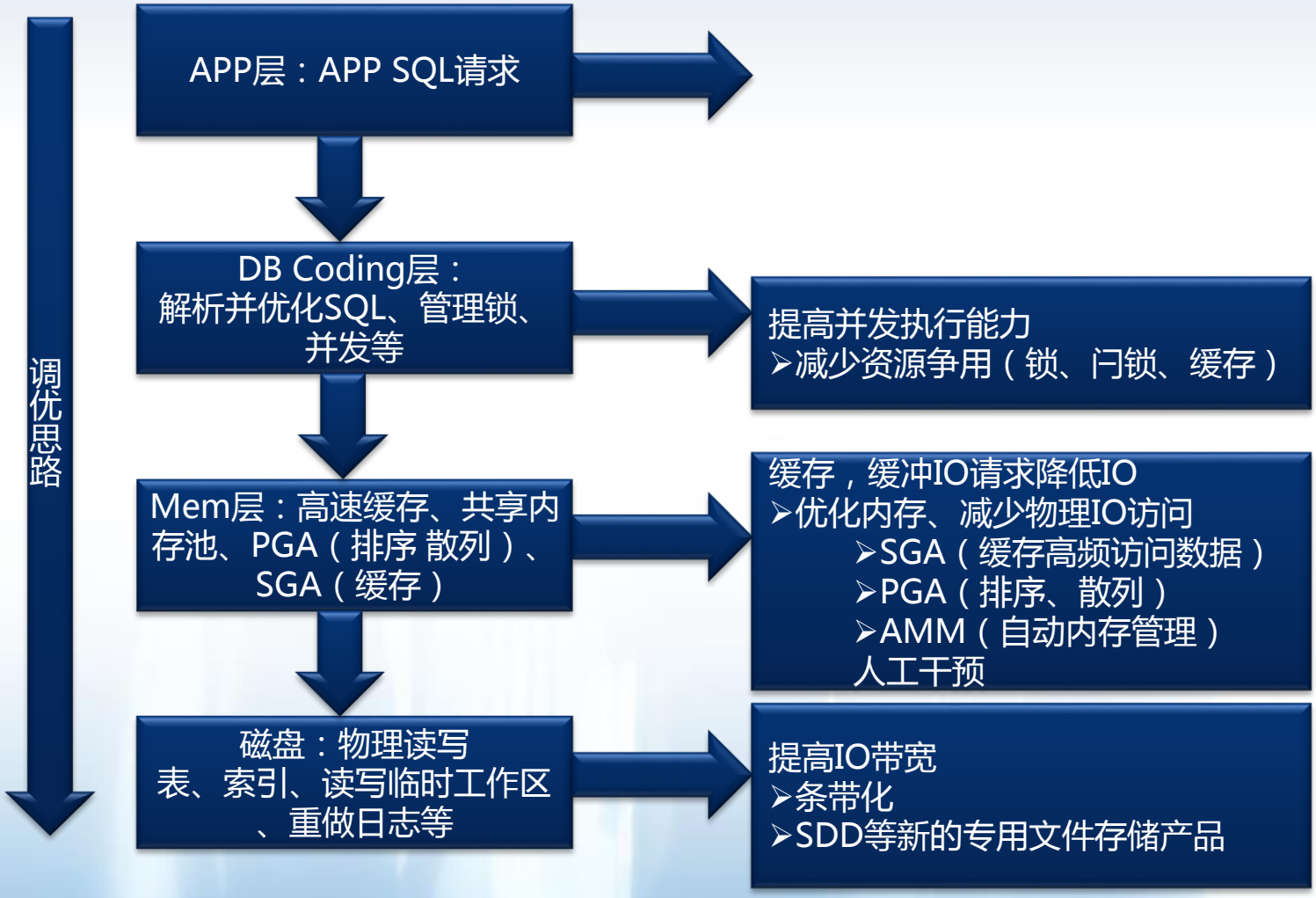
性能分析方向



存储体系


硬 盘
Bi(read)
Bo(write)
Bo(接近于0性能良好)
Bi(接近于0性能良好)
内 存
Free
Cache
Bi
Bo
虚 拟
内 存
Swap
SO（=0正常）
Si(=0正常)

DB优化思路


调优思路
APP层：APP SQL请求
DB Coding层：
解析并优化SQL、管理锁、
并发等
提高并发执行能力
➢减少资源争用（锁、闩锁、缓存）
Mem层：高速缓存、共享内
存池、PGA（排序 散列）、
SGA（缓存）
缓存，缓冲IO请求降低IO
➢优化内存、减少物理IO访问
➢SGA（缓存高频访问数据）
➢PGA（排序、散列）
➢AMM（自动内存管理）
人工干预
磁盘：物理读写
表、索引、读写临时工作区
、重做日志等
提高IO带宽
➢条带化
➢SDD等新的专用文件存储产品

APP优化思路

- 页面简化
- 表单压缩
- 局部刷新
- 仅取所需
- 减少不必要请求
- 逻辑清淅
- 谨慎继承
- 程序算法优化（二分查找　哈希表）
- 批处理
- 延迟加载
- 防止内存泄露
- 减少大对象引用
- 防止争用死锁

。。。。。

调优

- 何为调优：无非就是按照自己的意愿达到更加符合自己需求的结论。
  - 我们用的系统的性能是固定上限的
  - 调优只是接近理论最优值

调优策略

- 影响最大
- 调优最易
- 组合分析

影响最大调优

- 从绝大多数系统来说最容易出性能问题的是
  - 数据库上的SQL执行
  - 应用层或数据库上的CPU
  - 数据库上的IO

SQL调优

- 当我们出现SQL执行慢的问题怎么办？
  - 这是一个DBA的事情
  - 学会了解数据库是怎么执行SQL的
  - 了解数据库某些会变慢的原因
    - 锁
    - 索引

SQL执行计划1

- 在PLSQL中查看执行计划

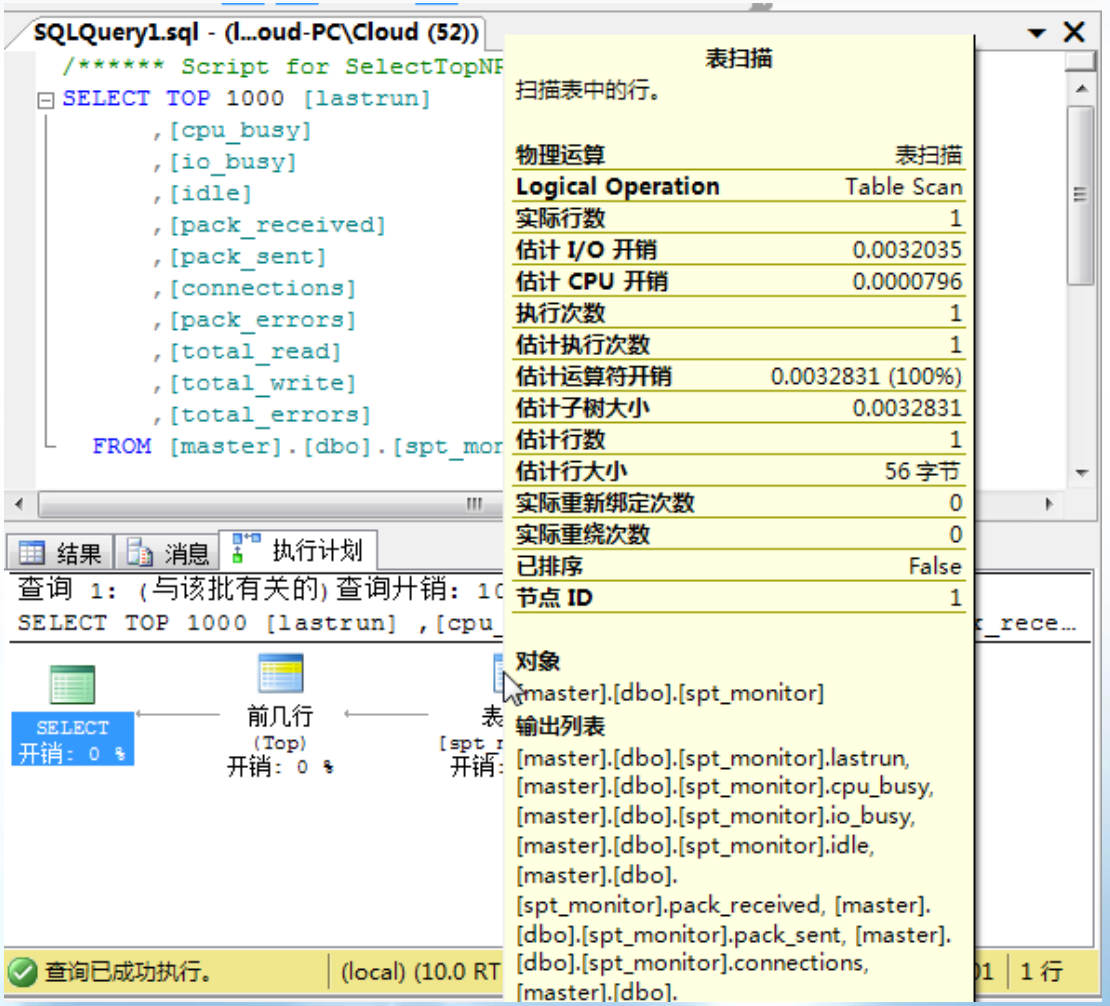
```
select * from sc where grade in(
  select grade from (
  select grade from sc order by grade desc) where rownum!=3) order by g
```

Optimizer goal: All rows

Description	Access predicates	Bytes	Cardinality	Cost	CPU cost	Depth	Distribution	Filter predicates	Id	IO cos
SELECT STATEMENT, GOAL = ALL_ROWS		134	2	8	11367265	0			0	6
SORT ORDER BY		134	2	8	11367265	1			1	6
FILTER						2		EXISTS (???	2	
TABLE ACCESS FULL		536	8	3	37287	3			3	3
FILTER						3		"GRADE"=:B1	4	
COUNT						4			5	
FILTER						5		ROWNUM<>3	6	
VIEW		104	8	4	5683553	6			7	3
SORT ORDER BY		104	8	4	5683553	7			8	3
TABLE ACCESS FULL		104	8	3	37127	8			9	3

SQL执行计划2

- 在MSSQL中查看执行计划



SQL执行计划3

- 在MySQL中查看执行计划
  - Mysql中并没有直接执行计划的概念，但是通过show profiling查询跟踪可以得到类似的信息。
  - Set profiling=1;//启动跟踪
  - Show profiles;//显示跟踪
  - Show profile for query 1;//详细跟踪编号为1的SQL语句
  - Show profile cpu for query 1;跟踪CPU

SQL分析调优1

- 通过跟踪获得数据库上慢的TOP SQL
- 检查这些SQL的执行计划，评估SQL慢的原因
- CPU？
- 内存？
- IO？
- 网络？

SQL分析调优2

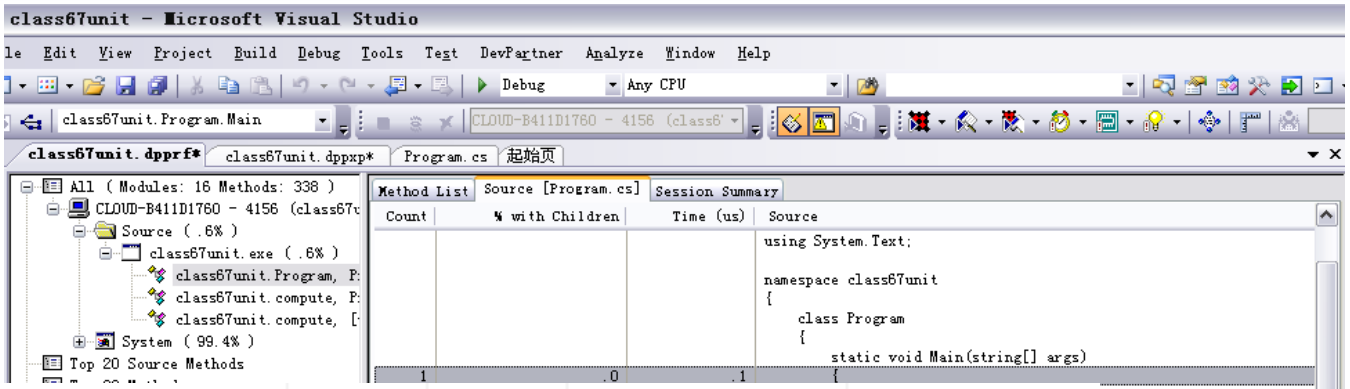
- 从SQL下手优化SQL结构
- 从索引或者视图下手减少查询的内容
- 从设计角度优化表结构
  - 数据分块
  - 范式
  - 业务转换

最易调优

- 往往调优最容易的地方反而是代码
  - 代码是最为成熟的部分
  - 代码主要开销的是CPU
  - 往往通过配置和应用平台的调整可以实现

代码执行跟踪

- 代码也可以跟踪来判断问题



Function	Self	Cum.	File	Called from
footer	0.12%	87.78%	/www/web/default/...	/www/web/default/p
require_once::/www/web/default/phpwind85/global.php	0.85%	8.39%	/www/web/default/...	/www/web/default/p
PrintEot	0.03%	0.47%	/www/web/default/...	/www/web/default/p
L::config	0.03%	0.27%	/www/web/default/...	/www/web/default/p
L::loadClass	0.05%	0.27%	/www/web/default/...	/www/web/default/p
L::reg	-	0.24%	/www/web/default/...	/www/web/default/p
S::gp	0.09%	0.21%	/www/web/default/...	/www/web/default/p
require_once::/www/web/default/phpwind85/template/...	0.09%	0.19%	/www/web/default/...	/www/web/default/p
L::style	0.03%	0.06%	/www/web/default/...	/www/web/default/p
S::gp	0.06%	0.06%	/www/web/default/...	/www/web/default/p

组合调优

- 让系统整理都没有明显瓶颈
  - 多少的表示层连接最合适
  - 多少的应用层
  - 多少的数据库
  - 发挥所有的资源是最有性价比的
  - 无节制的调优是毫无意义的

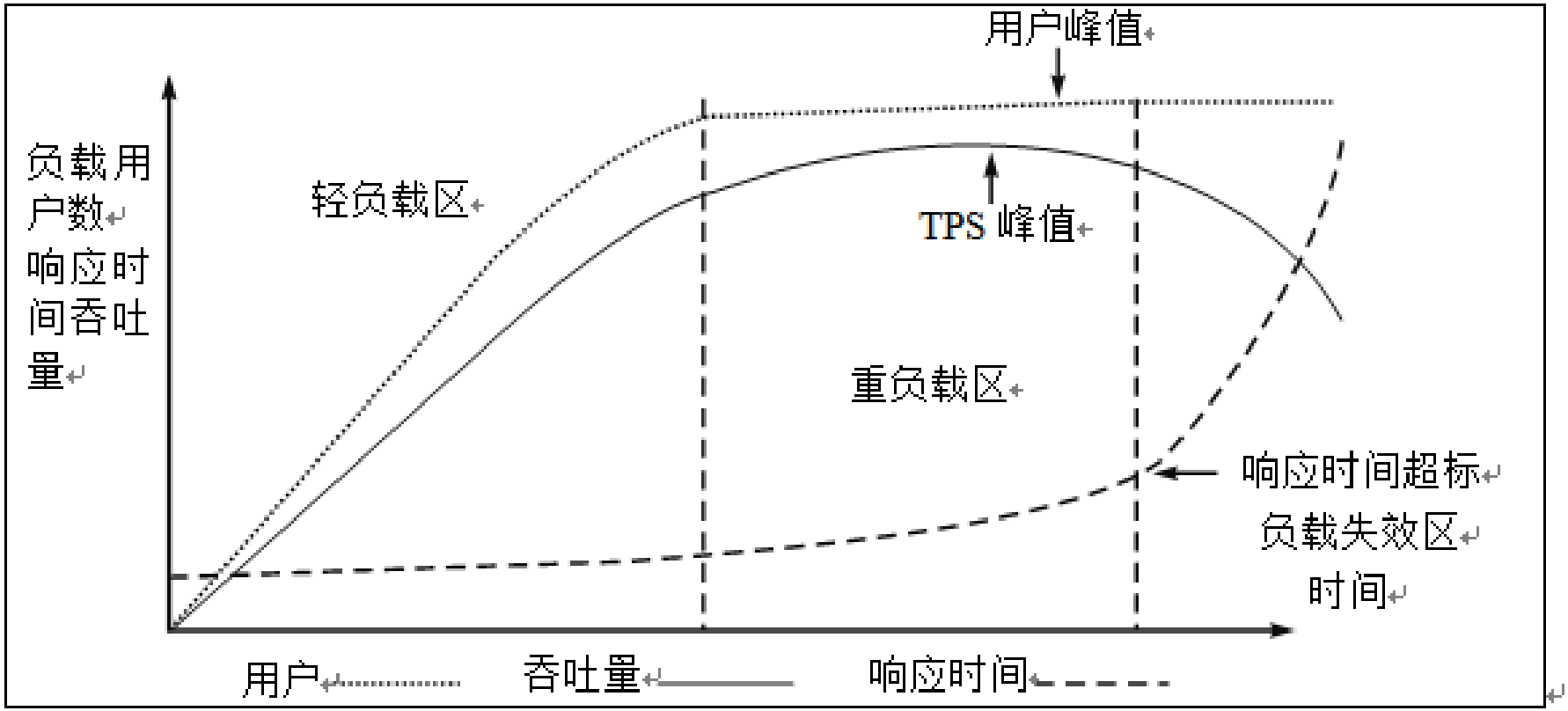
分析调优技巧1

- 通过理发师模型了解性能模型
  - 负载与响应时间的关系
  - 负载与吞吐量的关系
  - 负载状态的判断

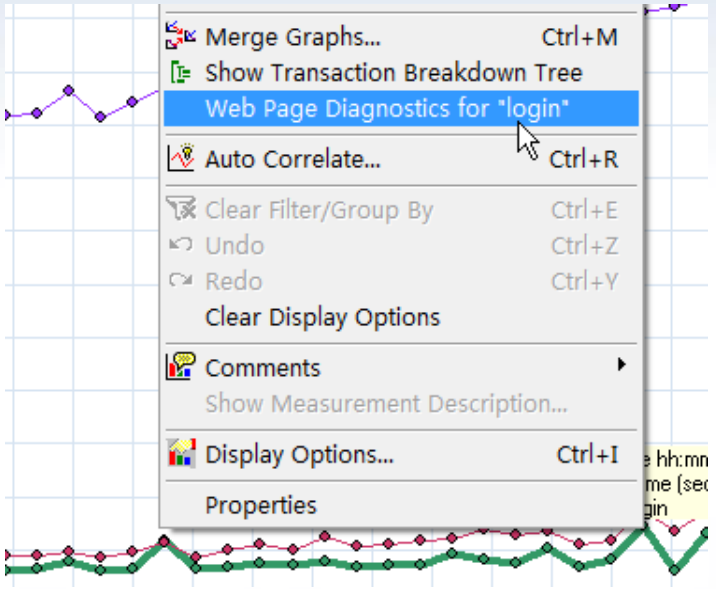
性能指标

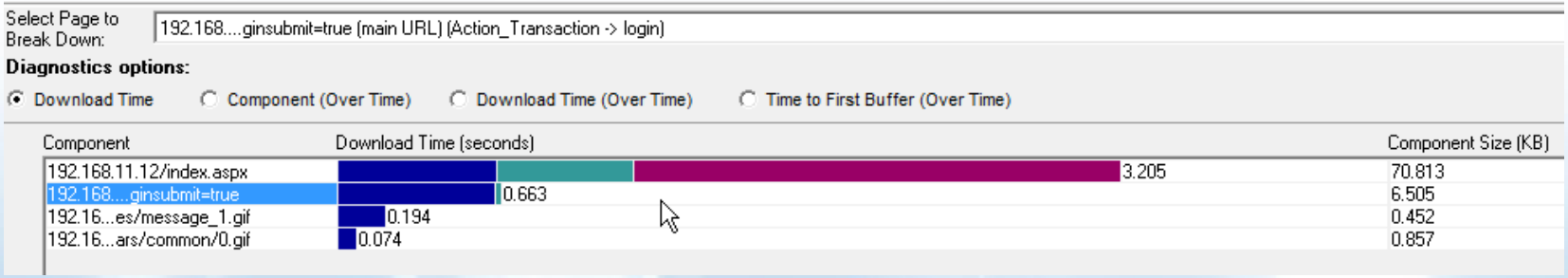
- 响应时间
  - 反映系统处理效率指标
  - 响应时间是从开始到完成某项工作所需时间的度量。在客户/服务器环境中，通常是从客户方测量响应时间。响应时间通常随负载的增加而增加。
- 吞吐量
  - 反映系统处理能力指标
  - 吞吐量是单位时间内完成工作的度量，在客户/服务器环境中通常是从服务器方进行评估。
  - 随着负载的增加，吞吐量往往增长到一个峰值后，然后下降，队列变长。在如客户/服务器这样的端到端系统中，吞吐量依赖于每个部件的运行。系统中最慢的点决定了整个系统的吞吐率。通常称此慢点为瓶颈。
- 资源利用率：反映系统能耗指标

负载模型


用户峰值
负载用户数
响应时间
吞吐量
轻负载区
TPS 峰值
重负载区
响应时间超标
负载失效区
时间
用户
吞吐量
响应时间

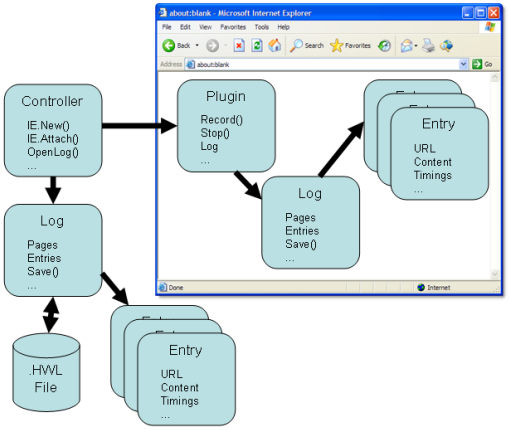
Webpagebreakdown



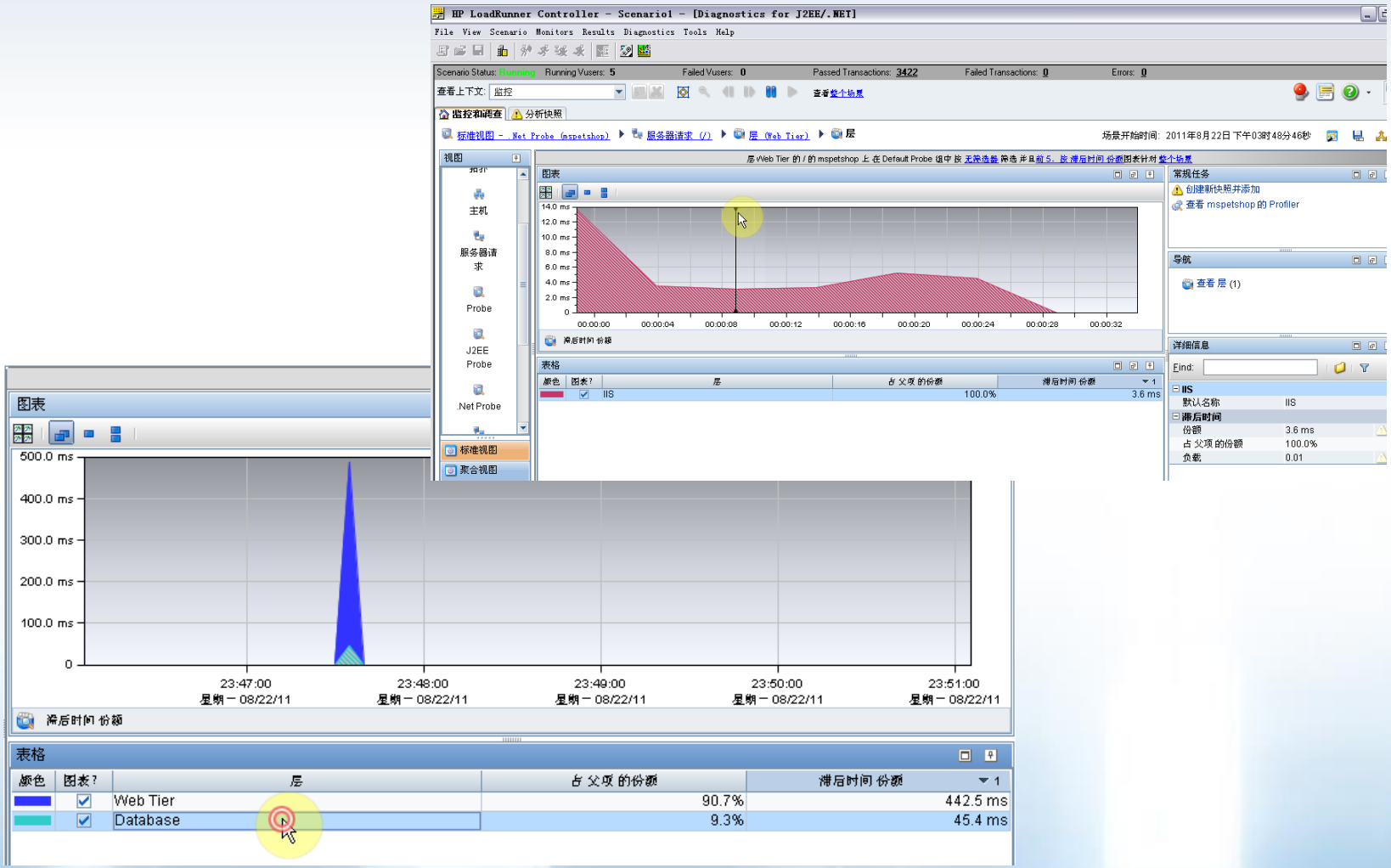


HTTPWATCH的整合

```
Dim filename
filename=FormatDateTime(now,2)&FormatDateTime(now,3)
filename=replace(filename,":","")
filename=replace(filename,"-","")
filename=replace(filename,"/","")
Dim control
Set control = CreateObject("HttpWatch.Controller")
Dim plugin
Set plugin = control.IE.New
plugin.Record
plugin.GotoUrl("http://www.baidu.com")
control.Wait plugin, -1

' Stop recording HTTP
plugin.Stop
' Close down IE
plugin.CloseBrowser
'msgbox filename
plugin.Log.Save("f:\\"&filename&".hwl")

set plugin=nothing
set control=nothing
```



Diagnostics Server



自定义手工探针

- 工具能够提供各种监控，我们自己也可以做
  - 数据在哪里
  - 如何获得
  - 如何加入性能测试工具
  - lr_user_data_point()函数的妙用

自定义手工探针案例


```
double atof (const char *string); /* Explicit declaration */
//关联函数获得处理时间保存在参数{disczp}内
//页面操作
lr_user_data_point("discuzprocess",atof(lr_eval_string("{disczp}")));
//将该数据写入discuzprocess自定义计数器
//atof将字符串转化为浮点型
//注意在使用atof函数时需要在脚本中添加该函数的扩展声明
```

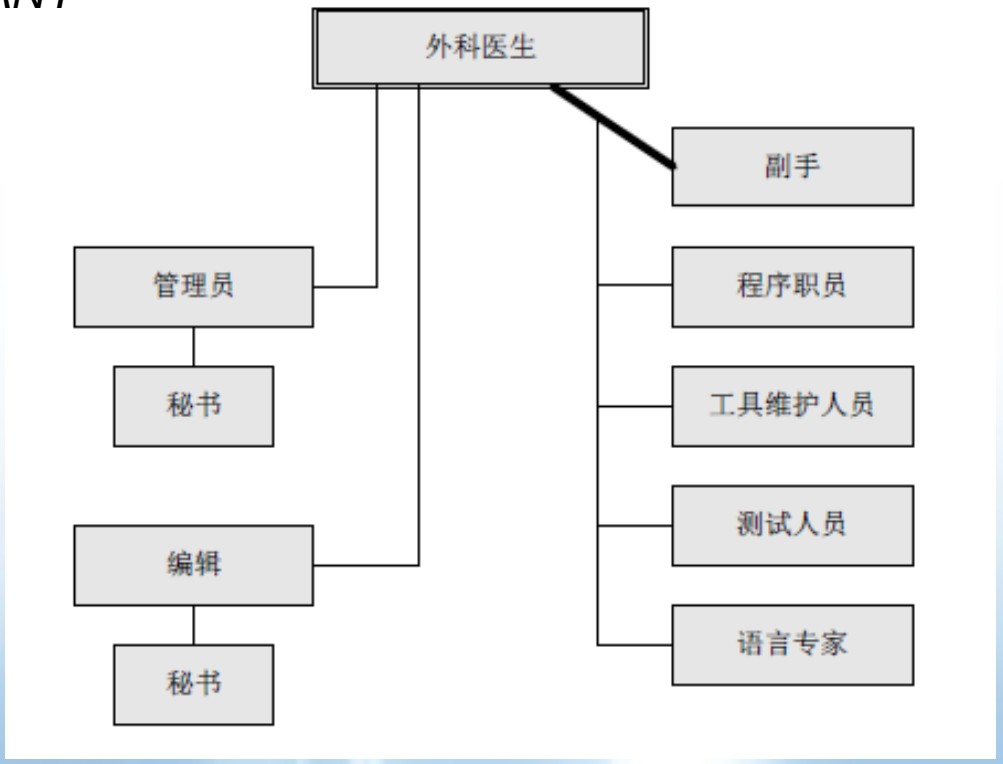
外科手术医生

这些研究表明，效率高和效率低的实施者之间具体差别非常大，经常达到了数量级的水平。

- SACKMAN, ERIKSON和GRANT[1]

These studies revealed large individual differences between high and low performers, often by an order of magnitude.

- SACKMAN, ERIKSON, AND GRANT[1]



从刘翔看调优

- 分析过程回顾
  - 八步上栏
  - 世界纪录
  - 伤退比赛
  - 七步上栏
- 测试结果
  - 12“87有遗憾么？
  - 对于自己比赛中的发挥，刘翔最满意的是前3个栏，不过他还是找出了自己的小问题，“只有在高强度、高水平的比赛中，才能发现自己的错误。”顺风尽管有助于跑出好成绩，但跨栏是讲究栏上平衡的技术性项目，所以刘翔认为，“风有点大，我被风推着跑，这使我在第三个栏到第六个栏之间出现了失误。”

调优水平

性能分析调优分为以下4个层次：

- 通过性能测试找到性能测试结果最好的版本。
- 通过性能测试找到各个版本的优缺点和瓶颈。
- 通过性能测试整合各个版本的优点，最终得到包含所有优点的版本。
- 通过各个阶段的调优，获得预估的最好性能理论最佳值，反复调优接近该目标。

相比HTTP，SPDY主要有三大改进:

- 多路复用请求。在一个SPDY连接内可以有无限个并行请求。TCP效率更高。
- 优化请求。客户端可以请求优先传递某些资源。在有高优先级的请求等待时，该功能可以避免让非关键资源占用网络通道的问题。
- 压缩页眉。现在的客户端以HTTP页眉的格式发送大量冗余数据。压缩这些页眉可以节省大量等待时间和带宽。

Step List

1. 需求篇
2. 分析篇*
3. 评审篇
4. 设计篇*
5. 准备篇
6. 实施篇
7. 收集篇
8. 分析篇*
9. 定位篇
10. 报告篇*

- 终于数据出来了，但是写好一个报告常常很困难
  - 不知道怎么写（格式问题）
  - 不觉得有什么可以写（分析问题）

报告格式

- 一个好的报告关键是包含下述内容
  - 清晰的测试目标
  - 准确的测试结果
  - 充分的测试环境准备
  - 详尽的测试数据记录及分析

性能测试报告分类

- 目标型
  - 强调系统是否满足了设计需求，用于验收
- 对比型
  - 强调配置下的基准测试，用于证明调优效果
- 分析型
  - 强调分析当前系统的问题，用于定位性能瓶颈

报告准则

1）脚本的工作原理应该完全相同

2）场景必须完全相同

3）每次对比测试的环境必须相同

报告要素

1）性能测试目的

看到一份报告，读者首先想了解的就是该文档的目的，这份报告到底有什么用，干了什么事，得出了什么样的结论。快速、全面、清晰地了解该文档编写的目的和报告的结论，是一份好的性能测试报告应该具备的要素。

2）测试环境

性能测试是基于系统的，所以必须要说明整个性能测试基于的软硬件环境，来确保性能测试的数据有一定的普遍代表性。作为一次性能测试，除非为了做配置测试，否则不建议做任何测试环境的调整，而只能改变被测系统的版本。

3）测试工具及测试方法的说明

虽然我们一直在谈如何使用LoadRunner来进行性能测试，但这里还是再要强调一次，性能测试不是LoadRunner，只是LoadRunner比较流行或者使用起来相对方便。在这次性能测试中，我们通过对工具的分析和比较，才确定使用LoadRunner。这里需要对该工具进行简单的说明，而对这次测试所编写的测试代码需要进行比较详尽的说明，让读者了解这次性能测试的原理和负载规则。

4）测试结果数据列表

通过运行场景，得到相关数据，然后统一生成数据的表格和图形，方便用户能够直观地看出多次测试结果数据的差异。

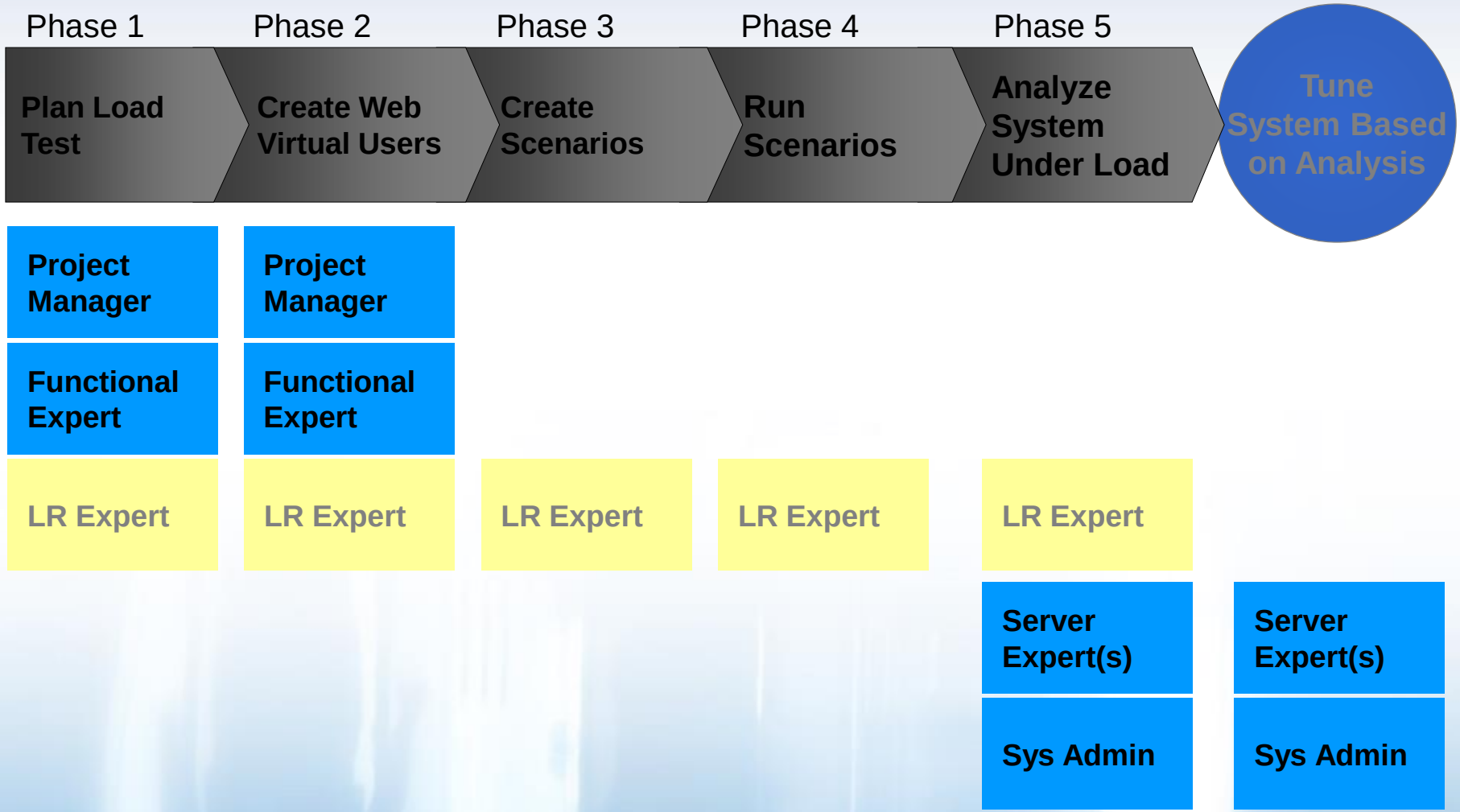
5）比较分析

在上面的数据中，将数据差异的原因进行分析，得出产生差异的可能原因。

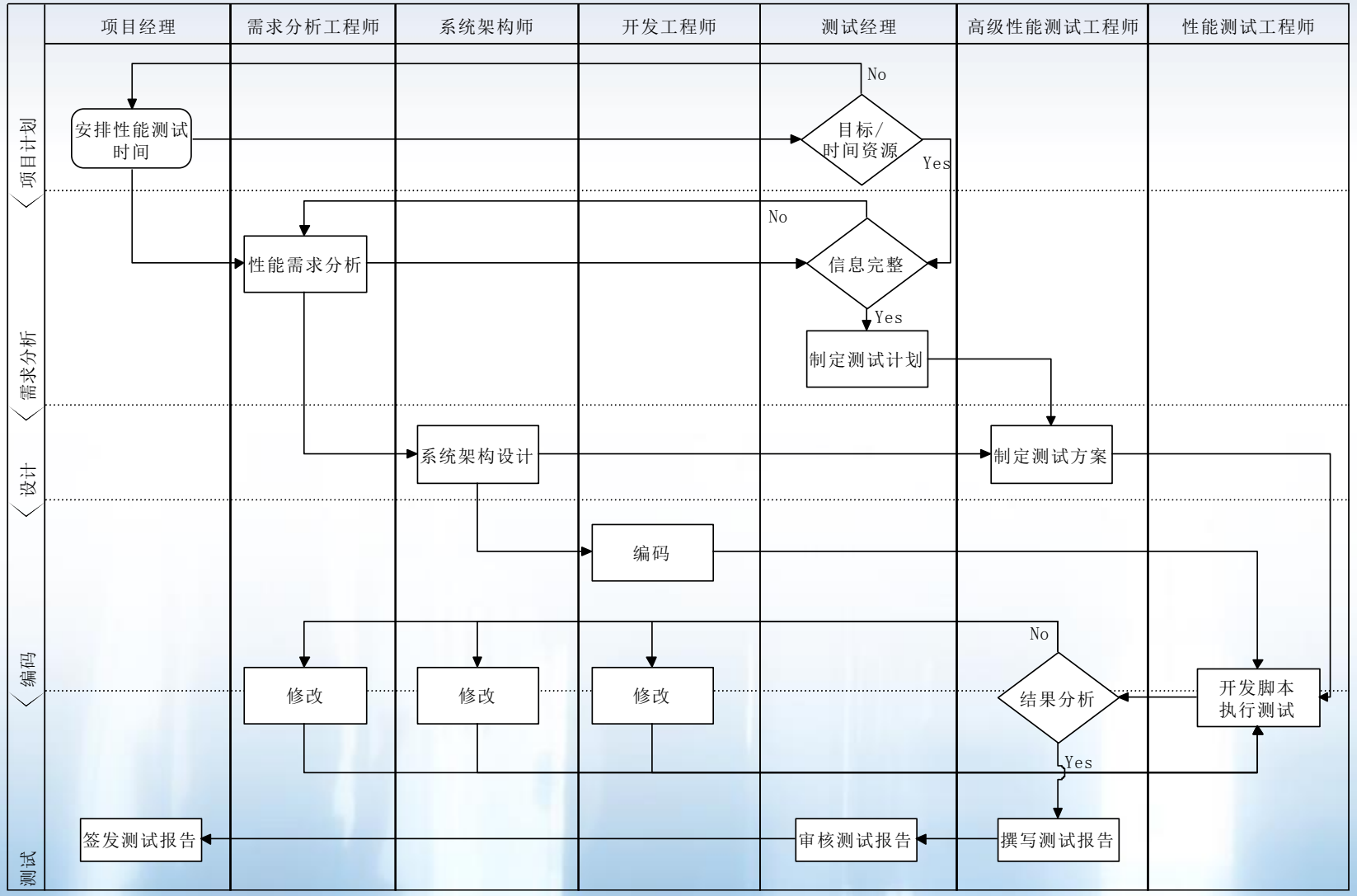
6）总结性能测试结论

总结所有的数据和内容，给出最后的定论，完成性能测试报告。

LoadRunner 团队职责



性能测试流程


项目经理
需求分析工程师
系统架构师
开发工程师
测试经理
高级性能测试工程师
性能测试工程师
项目计划
需求分析
设计
编码
测试
安排性能测试时间
目标/时间资源
No
Yes
性能需求分析
信息完整
No
Yes
制定测试计划
系统架构设计
制定测试方案
编码
修改
修改
修改
结果分析
No
Yes
开发脚本执行测试
签发测试报告
审核测试报告
撰写测试报告

性能测试主要角色及职责

- 项目经理
  - 计划测试时间，监督项目进度
    - 1、项目经理自己了解性能测试，进行合理的性能测试时间安排；
    - 2、通过“进度”度量获得项目经验数据，据此做出正确的时间安排；
    - 3、指定测试经理根据项目进度，安排性能测试进度；

- 需求分析工程师
  - 撰写性能测试需求
    - 1、用户可能不能明确提出性能方面的需求，需求分析工程师需要指导用户确定性能需求
      - A、系统用户数
      - B、在不同用户数量级别的并发用户数下，系统的响应时间和服务器的资源利用率；
      - C、系统的处理能力；

性能测试主要角色及职责

- 系统架构师
  - 根据需求做出正确的系统架构的设计
- 开发工程师
  - 根据架构设计的要求进行编码
- 测试经理
  - 制定并组织评审性能测试计划
  - 组织资源
  - 跟踪项目进度
  - 处理性能测试过程中遇到的各种问题

性能测试主要角色及职责

- 高级性能测试工程师
  - 制定性能测试方案
  - 分析测试结果
- 性能测试工程师
  - 开发Vuser Script
  - 执行性能测试场景
  - 提交性能测试结果
  - 执行回归测试

配置管理

- Performance Center和Quality Center
- 管理性能测试便于基准和配置测试
- 基准测试（ Benchmark Testing）：
  - 在一定的软件、硬件及网络环境下，模拟一定数量虚拟用户运行一种或多种业务，将测试结果作为基线数据，在系统调优或者系统评测过程中，通过运行相同的业务场景并比较测试结果，确定调优是否达到效果或者为系统的选择提供决策数据。
- 配置测试（Configuration Testing）：
  - 在不同的软件、硬件以及网络环境配置下，通过运行一种或多种业务在一定的虚拟用户数量情况下，获得不同配置的性能指标，用于选择最佳的设备及参数配置。

VUGEN代码结构

- Scenario.usr文件是VuGen打开脚本的主文件，文件内包含了一些VuGen的通用设置和运行属性。Default.cfg和default.usp是配置文件，其中runtime setting存放在default.usp文件中

  - Action.c
  - Vuser_end.c
  - Vuser_init.c
  - Globals.h
  - Default.cfg
  - Default.usp
  - Scenario.usr

VUGEN运行

- 运行后，脚本目录下会生成result1目录，存放对运行时的测试结果记录。Mdrv.log和output.txt文件是运行脚本所产生的log文件。Mdrv_cmd.txt是命令行启动mdrv的参数写法，三个.c文件是运行脚本所需要的系统函数

  - Combined_scm_.c
  - Scm.c
  - Pre_cci.c
  - Logifle.log
  - Mdrv.log
  - Mdrv_cmd.txt
  - Options.txt
  - Output.txt

使用SVN管理性能测试

- 测试脚本的管理
- 测试结果的管理
- 性能测试的自动化

（1）确保当前目录为工作目录，同时SVN命令在系统PATH环境变量中。

（2）使用SVN 的cp参数创建Tag。

（3）通过Svn checkout命令检出Tag。

（4）完成版本发布记录

（5）运行场景。

（6）使用Svn add命令将性能测试数据提交至服务器。

测试环境的自动化部署

- Linux Shell
- VBS

```
svn up /dailybuild/svn
for filename in `ls -t /dailybuild/svn/tags/`
grep $filename /dailybuild/build.log  1>/dev/null 2>&1
if [ $? -eq 1 ]
then
echo "find new version $filename"
rm -rf /dailybuild/build/*.*
cp /dailybuild/svn/tags/$filename /dailybuild/build/
echo "build new version....."
cd build
tar zxvf /dailybuild/build/$filename
echo $filename `date` | tee -a /dailybuild/build.log 1>/dev/null 2>&1
else
echo "no new version need build"
fi
```

场景执行

Wlrun.exe -TestPath g:\Scenario\scenario1.lrs –Run
上面这个命令说明调用wlrun.exe启动场景，通过-TestPath参数启动g:\scenario\ scenario1.lrs场景文件，-Run参数说明启动该场景。

命令参数	说　明
TestPath	Path to the scenario, for example, 这里可以填写场景的物理地址 C:\LoadRunner\scenario\Scenario.lrs This argument can also be used for a scenario residing in a Quality Center database. For example, 也可以填写场景在TD/QC上的地址 "[TD]\Subject\LoadRunner\Scenario1" If the path includes blank spaces, use quotation marks. 如果路径中存在空格，那么使用引号
Run	Runs the scenario, dumps all output messages into res_dir\output.txt and closes Controller 运行场景，场景结束后将所有信息输出到以下地址并关闭Controller
InvokeAnalysis	Instructs LoadRunner to invoke Analysis upon scenario termination. If this argument is not specified, LoadRunner uses the scenario default setting. 当场景结束时自动调用Analysis，如果该参数没有被定义，则使用场景中的默认设置
ResultName	Full results path. 自定义场景结果的完整地址（包括路径和名称） For example, "C:\Temp\Res_01"
ResultCleanName	Results name. 自定义结果的名称 For example, "Res_01"
ResultLocation	Results directory. 自定义结果的路径 For example, "C:\Temp"

性能测试总结1

- 从制动来重谈性能测试分析及调优
- 性能测试要尽早进行
- 分析的层次
  - 硬件
  - 系统
  - 服务/框架
  - 代码

性能测试总结2

- 用事实证明推测
- 收集足够的证据才对系统进行优化
- 优先验证简单的假设
- 日志文件并不能看到所有的问题
- 层层剥离缩小范围
- 分割单元定位细节

项目实践

- 如何做需求
- 如何做计划
- 如何做方案
- 如何搭环境
- 如何开发脚本
- 如何监控
- 如何负载
- 如何定位
- 如何调优
- 如何写报告

性能测试工具

- 性能测试工具原理
- 性能测试技术
  - 参数化
  - 集合点
  - 关联
  - 事务
  - 代码调试

HTTP协议简介

- HTTP协议的主要特点可概括如下：
  - 1、支持客户/服务器模式；
  - 2、简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快；
  - 3、灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记；
  - 4、无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间；
  - 5、无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

HTTP请求

GET /WebTours/ HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-ms-application, application/vnd.ms-xpsdocument, application/xaml+xml, application/x-ms-xbap, application/x-shockwave-flash, application/x-silverlight, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*

Accept-Language: zh-cn

UA-CPU: x86

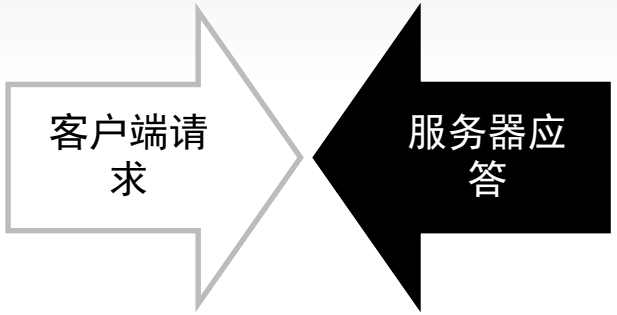
Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; CIBA; .NET CLR 3.5.30729; .NET CLR 3.0.30618; InfoPath.2)

Host: 127.0.0.1:1080

Connection: Keep-Alive

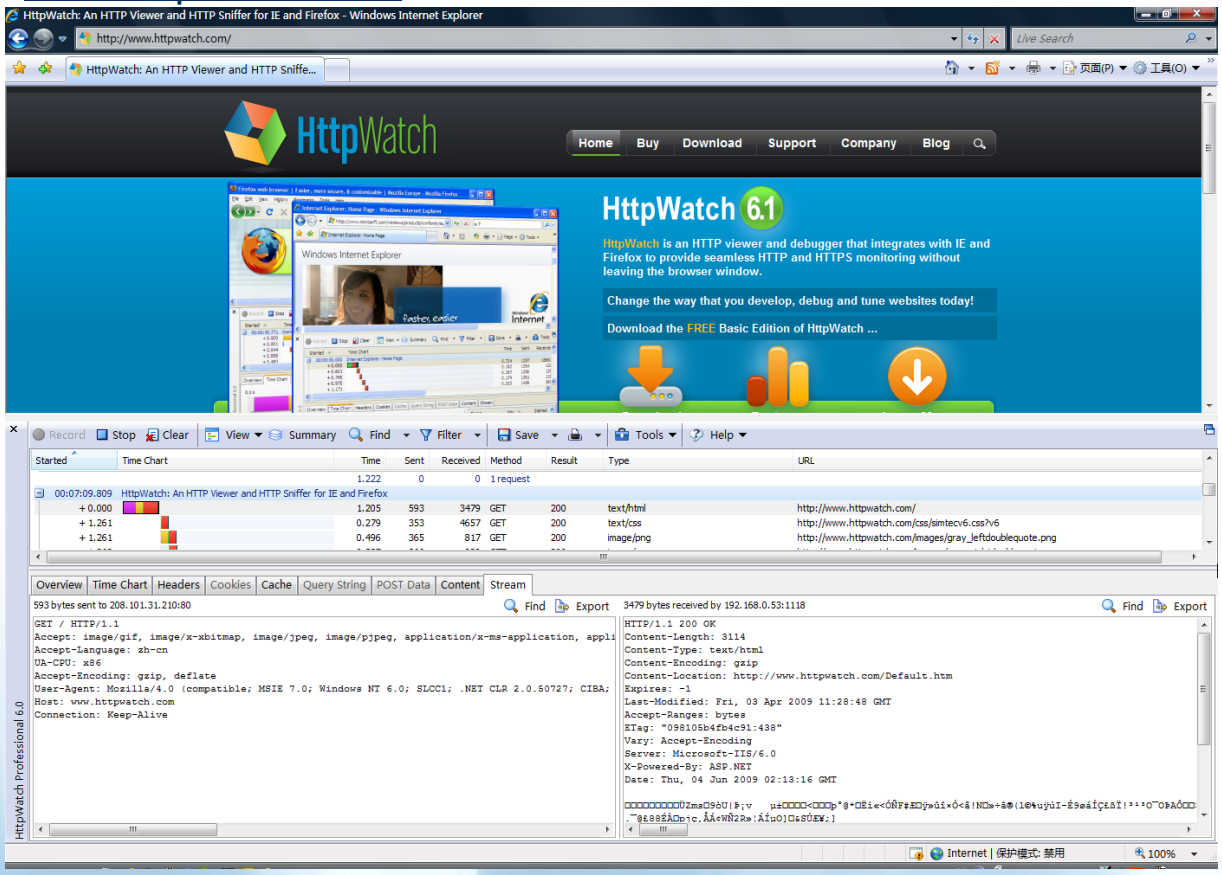
HTTP应答



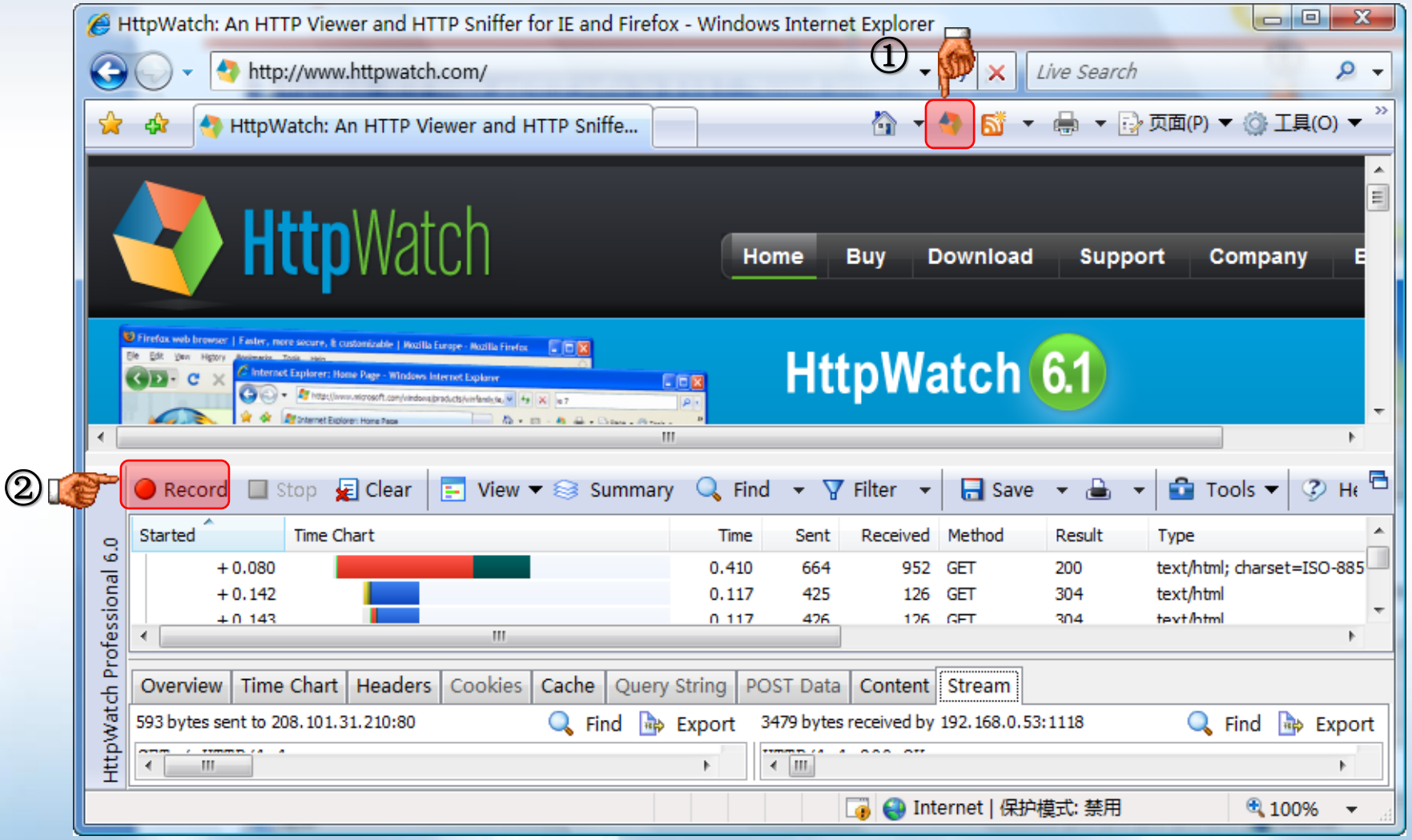

```
HTTP/1.1 200 Ok
Server: Xitami
Date: Thu, 04 Jun 2009 02:06:18 GMT
Content-type: text/html
Content-length: 322
Last-modified: Tue, 01 Jan 2008 19:53:26 GMT
<HTML>
<HEAD>
<title>Web Tours</title>
<frameset rows = "65,*" border=1 bordercolor=#E0E7F1>
<frame name="header" src=header.html scrolling=no noresize marginheight=2 marginwidth=2>
<frame name="body" src=welcome.pl?signOff=true scrolling=auto noresize marginheight=2
      marginwidth=2>
</frameset>
</head>
</html>
```

HttpWatch工具介绍

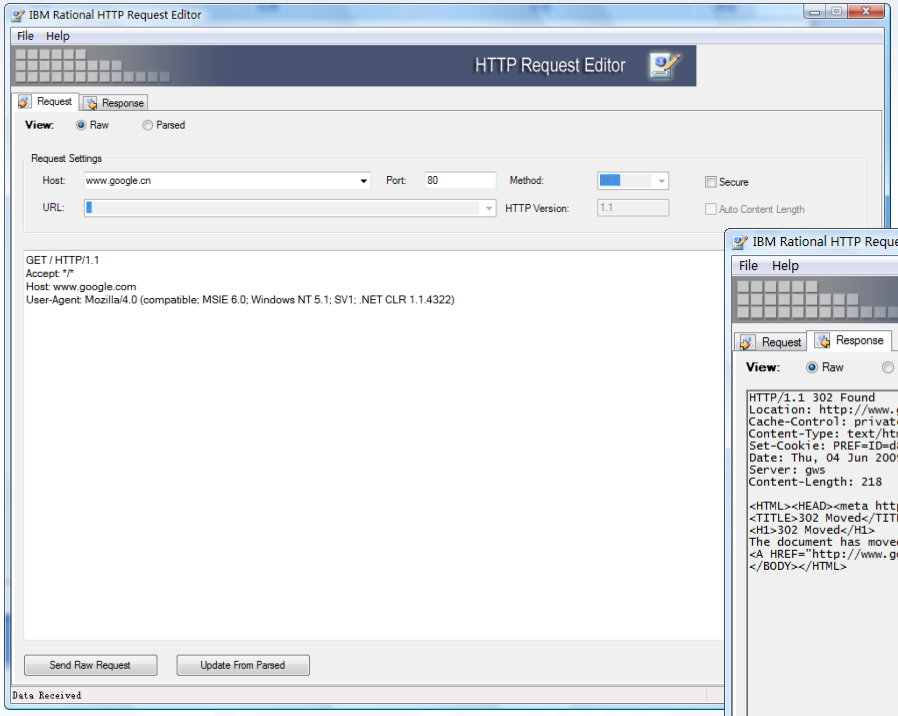
- HttpWatch is an HTTP viewer and debugger that integrates with Internet Explorer to provide seamless HTTP and HTTPS monitoring without leaving the browser window.
（URL：www.httpwatch.com）



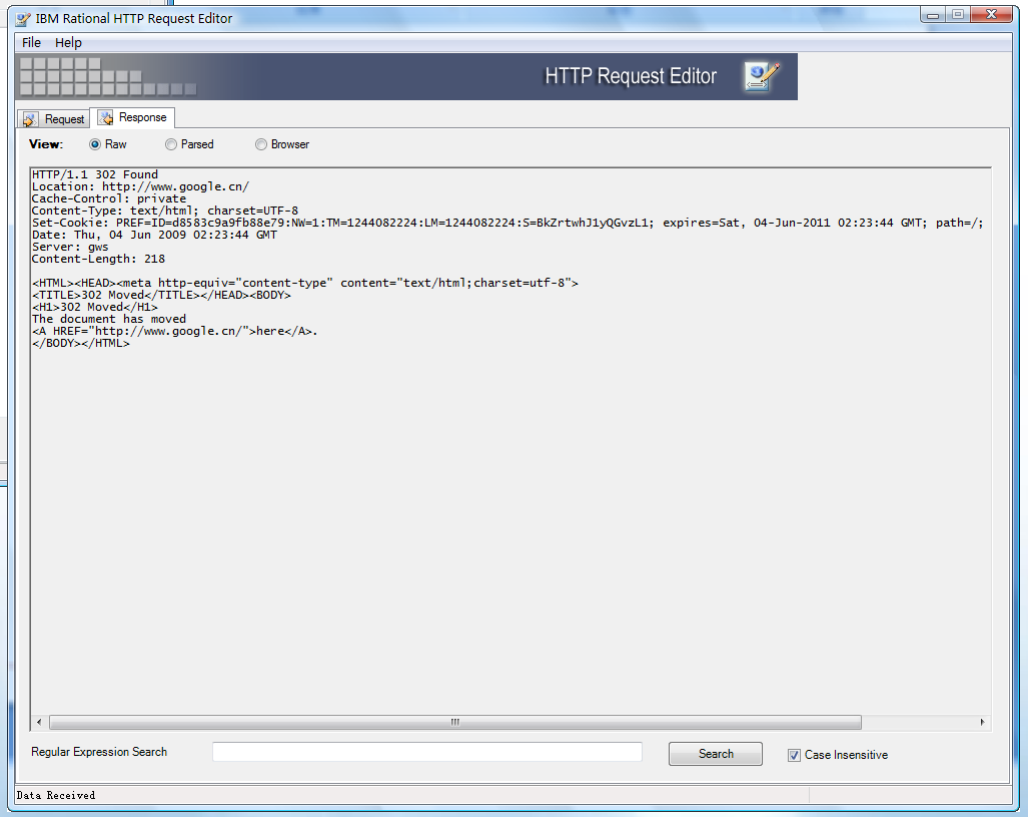
HttpWatch使用



HTTP Request Edit



模拟请求发送



服务器返回

WEB_LINK与WEB_URL

- Web_link如何处理链接
- Web_link如何处理同名链接
- Web_url如何应用

- 何时使用Web_link何时使用Web_url
- Web_custom_request函数能做啥

参数与变量

- 定义
  - 参数定义
    - lr_save_string(“51testing”,”param”);
  - 变量定义
    - Int x;//遵守C语言规范
- 调用方法
  - 参数需要双引号
    - lr_eval_string(“{param}”);
  - 变量直接调用，不能使用双引号，否则做字符串处理
- 作用域
  - 参数为全局
  - 变量默认为局部，可以定义全局变量

XML的处理

可扩展标记语言 (Extensible Markup Language, XML)，用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。 XML是标准通用标记语言 (SGML) 的子集，非常适合 Web 传输。

生成XML


```
Option Explicit
Const  ForReading = 1
Const  ForWriting = 2
Const  ForAppending = 8
Dim fso,  file, file1, msg,outmsg
Set  fso = CreateObject ("Scripting.FileSystemObject")
Set  file =  fso.OpenTextFile ("input.xml",ForReading)
Set  file1 =  fso.OpenTextFile ("output.xml",ForWriting)
Do While Not  file.AtEndOfStream
msg =  file.ReadLine
msg = replace(msg,"""","\""",1,-1,1)
outmsg= """"&msg&""""
file1.WriteLine outmsg
Loop
file1.Close
file.Close
Set file1=nothing
Set file = Nothing
Set fso = Nothing
```

查找XML


```
<?xml version="1.0" encoding="GB2312"?>
<bookstore>
      <book>
   <id>No1</id>
      <title>性能测试进阶指南</title>
      <author>cloud</author>
      <year>2010</year>
      <price>45.0</price>
      </book>
      <book>
   <id>No2</id>
      <title>性能测试进阶指南2</title>
      <author>chen</author>
      <year>2012</year>
      <price>79.0</price>
   </book>
</bookstore>
```

LR_GET_XML_VALUES


```
lr_xml_get_values("XML={xmlparam}",
"FastQuery=/bookstore/book/title",
"ValueParam=title",
LAST);

gvcount=lr_xml_get_values("XML={xmlparam}",
"Query=/bookstore/book/title",
"ValueParam=title",
"SelectAll=yes",
"NotFound=continue",
LAST);
```

LR_XML_FIND


```
lr_xml_find("XML={xmlparam}","Query=/bookstore/book/id","Value=No*.","SelectAll=yes","UseRegExp=yes",LAST);
```

- 参数数组必须满足以下两个条件：
  - 参数必须都是以相同的名字开头的，后接下画线加数字的方式顺序赋值。
  - 参数数组必须有一个“参数名_count”的参数来记录数组的长度。

参数数组的处理

- lr_paramarr_idx()

```
char * siteval;
siteval = lr_paramarr_idx("website", 2);
```

- lr_paramarr_len()

```
int arrSize;
arrSize = lr_paramarr_len("website");
```

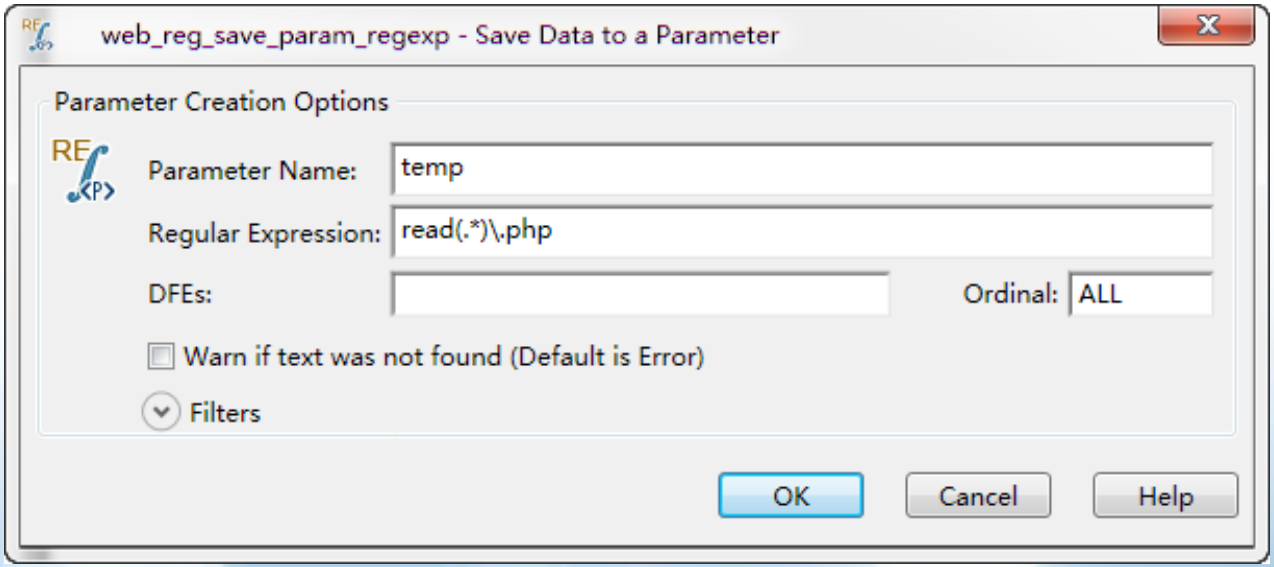
- lr_paramarr_random()

```
char * siteval;
siteval = lr_paramarr_random("website");
```

正则表达式关联

在LR11中提供了web_reg_save_param_regexp正则表达式关联

需要关联返回的内容需要用（）圆括号标记。例如，这里的read(.*)\.php就是指所有符合read开头.php结尾中间的任何内容都关联保存到参数temp中，这里的\是转义符，确保.号能够正确地当做普通字符来匹配。



正确计算Wasted Time

```
Action()
{
    int i;
    int baseIter = 100;
    char dude[1000];
    merc_timer_handle_t timer;
    // Examine the total elapsed time of the action
    //Start transaction
    lr_start_transaction("Demo");
    timer=lr_start_timer();
    for (i=0;i<=baseIter*1000;i++) {
        sprintf(dude,"This is the way we waste time in a script = %d", i);
      }
wasteTime=lr_end_timer(timer);
    lr_wasted_time(wasteTime*1000);
    lr_end_transaction("Demo", LR_AUTO);
    return 0;
}
```


事务时间
函数自身百分之1秒误差
thinktime
Wasted time
响应时间
网络延迟
连接服务器
发送数据
等待服务器处理
收回数据
WWW
APP
DB
连接
Web机制
表单处理
https
方法
线程池
JDBC
连接
缓存池检查
SQL
数据返回

用好analysis

- 看懂图
- 做好Merge
- 用好Webpagebreakdown
- 做好Auto Correlate

对Jar进行性能测试

- 用LR对Jar进行性能测试
- 用junitperf进行性能测试

用LR测JAR

- 在Eclipse中调用Jvuser
- 载入Jar包
- 编写调用代码
- 负载得到数据

用junitperf测JAR

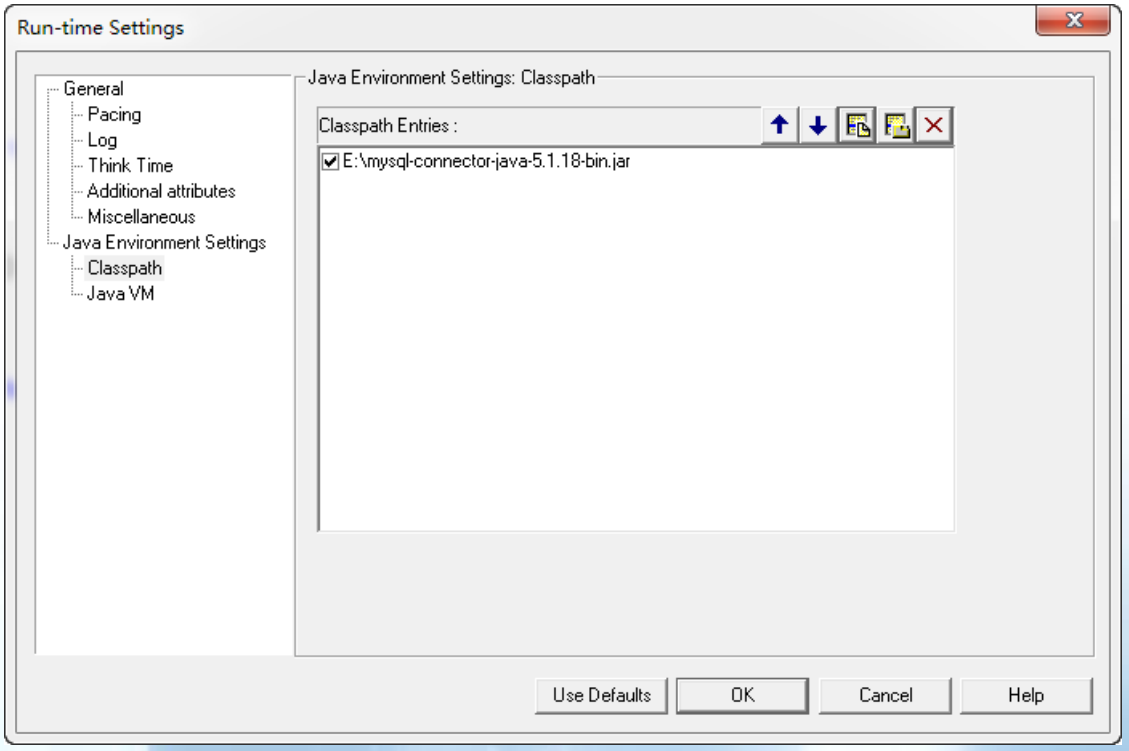

```
public static Test suite() {
long maxElapsedTime = 2000;
Test testCase = new computetest("testAdd");
Test timedTest = new TimedTest(testCase,
maxElapsedTime);
return timedTest;
}

public static void main(String[] args) {
junit.textui.TestRunner.run(suite());
}
```

Junitperf负载


```
public static Test Loadsuite(){
int users = 10;//模拟负载用户数
int iterations = 10;//每个用户数的调用次数
long maxElapsedTime = 20000; //超时上限
//Timer timer = new ConstantTimer(1000);
Timer timer = new ConstantTimer(100);//线程启动间隔时间100毫秒
Test testCase = new computetest("testSum");
Test loadTest = new LoadTest(testCase, users, iterations, timer);
Test timedTest = new TimedTest(loadTest, maxElapsedTime);
return timedTest;
}
```

Java Vuser测试MySQL数据库性能

- 通过JDBC编写一个连接MySQL数据库查询的Java Vuser脚本，用来测试数据库的性能，同样的写法可以应用在SQL Server或Oracle中。
- 这里要使用MySQL JDBC驱动程序mysql-connector-java-*.jar，并加入到ClassPath中。



代码


```
import lrapi.lr;
import java.sql.DriverManager;
import java.sql.*;
import com.mysql.jdbc.Connection;

public class Actions
{

public int init() throws Throwable {
        return 0;
}//end of init
```

```
public int action() throws Throwable {
        int ColumnCount;
        int RowCount;
        String driver = "com.mysql.jdbc.Driver";
        String url = "jdbc:mysql://localhost:3306/Phpwind85";
      String user = "root";
      String password = "";
      try {
      Class.forName(driver);
      lr.start_transaction("jdbc");
      Connection conn = (Connection) DriverManager.getConnection(url, user, password);
      if(!conn.isClosed())
       System.out.println("数据库连接成功！");
       Statement stat=conn.createStatement();

    lr.start_transaction("search");
```

```
ResultSet rs = stat.executeQuery("select * from pw_members");

  lr.end_transaction("search", lr.AUTO);

      ResultSetMetaData rsmd = rs.getMetaData();
      ColumnCount = rsmd.getColumnCount();
      rs.last();
      RowCount=rs.getRow();
      System.out.println("结果集的列数:" + ColumnCount);
      System.out.println("结果集的行数:" + RowCount);
      rs.close();
      conn.close();

                        lr.end_transaction("jdbc", lr.AUTO);

        }
```

```
 catch(ClassNotFoundException e) {
                System.out.println("找不到驱动程序");
                e.printStackTrace();
          }
      catch(SQLException e) {
                e.printStackTrace();
           }
                    return 0;
          }//end of action


          public int end() throws Throwable {
                    return 0;
          }//end of end
}
```

感谢您的关注