# 课程介绍

* 1. 逻辑控制器
* 2. Jmeter各元件总结（掌握）
* 3. Jmeter性能测试（掌握）

1. 逻辑控制器
   1. 什么是逻辑控制器

Jmeter逻辑控制器是可以控制采样器(Sampler)的执行顺序，它由多个逻辑控制语句封装成不同功能的组件组成。只对其子节点的sampler有效。

jmeter提供了多种逻辑控制器，它们各个功能都不尽相同，大概可以分为2种使用类型：

①.控制测试计划执行过程中节点的逻辑执行顺序，如：Loop Controller（循环控制器）、If Controller（如果if控制器）等；

②.对测试计划中的脚本进行分组，方便JMeter统计执行结果以及进行脚本的运行时控制等，如：Throughput Controller（吞吐量控制器）、Transaction Controller（事务控制器）等

* 1. 常用的逻辑控制器

1. 如果（If）控制器

2. ForEach控制器

3. 循环控制器

* + 1. 如果（If）控制器

作用：条件成真,则执行控制器下所有取样器

使用方法及步骤：

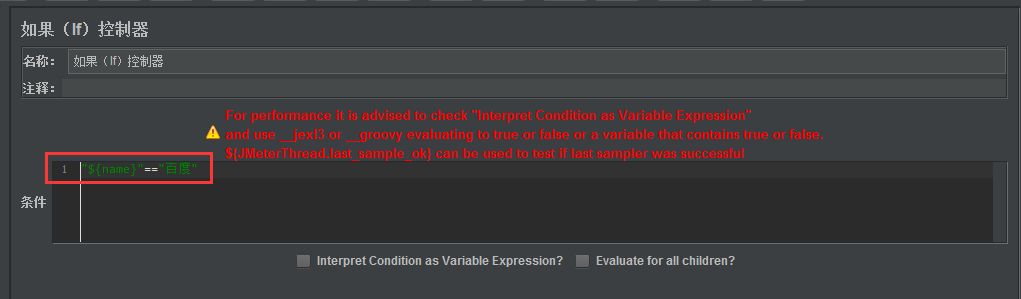
1. 测试计划->线程组->用户自定义变量

2. 线程组->如果（If）控制器

3. 如果（If）控制器->HTTP请求

4. 测试计划-察看结果树

If控制器配置



if控制器参数:

**名称** - 用于标识JMeter元素树中的元素的控制器

**注释** - 可用于包含指定条件的描述性解释的字段（有时条件可能非常复杂，将此类注释留作引用很有用）

**条件（默认Javascript）** - 由执行流程验证的条件，用于决定是否应执行子元素。默认情况下，条件被解释为返回“true”或“false”的Javascript代码

**将条件解释为变量表达式？** - 此参数适用于不需要评估Javascript代码的情况。默认方法接受并将指定的条件解释为Javascript代码，然后验证结果是否等于true或false。但是如果您选择此参数，则不会使用任何Javascript解释，并且条件将被视为JMeter变量

**评估所有孩子？** - 如果选择此属性，则将检查每个子条目的指定条件，而不仅仅是一次，因为默认情况下已完成

注意事项:

* + - 1. JMeter If Controller使用Javascript进行条件解释
      2. 如果要比较两个字符串，则必须在引号中指定它们（即使对于变量也应使用引号）。
      3. 尽可能使用'解释条件作为变量表达式',运行模拟大量用户的性能脚本时，应记住用于测试的资源数量。使用If Controller时，请记住默认情况下，If Controller使用Javascript解释条件，并且每个解释都占用资源。因此，如果你可以使用“解释条件作为变量表达式”而不是默认的Javascript解释,它节省了性能资源。
      4. 如果有一个无法使用变量表达式解释器执行的复杂条件，则最好使用Groovy和JEXL解释器,即使用\_\_jexl3或\_\_groovy函数[。](https://jmeter.apache.org/usermanual/component_reference.html#If_Controller)
      5. 使用$ {JMeterThread.last\_sample\_ok}检查交易请求

$ {JMeterThread.last\_sample\_ok}函数返回上一个请求采样器的结果，而'评估所有孩子？' 参数告诉JMeter验证针对每个子请求采样器的条件。

因此，如果先前的请求成功，则$ {JMeterThread.last\_sample\_ok}将为下一个请求返回“true”，它将运行。否则，$ {JMeterThread.last\_sample\_ok}将返回false，并且不会运行所有剩余的请求。

* + 1. foreach控制器

作用：ForEach控制器一般和用户定义的变量一起使用，在用户自定义变量中读取一系列相关的变量。

**使用方法及步骤：**

1. 测试计划->线程组

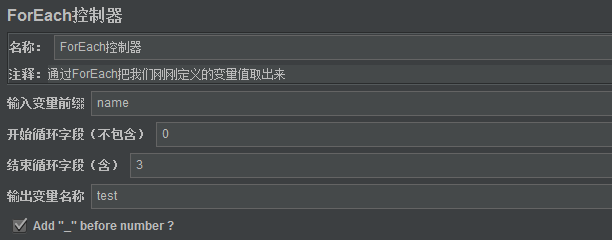
2. 线程组->用户定义的变量

3. 线程组->ForEach控制器

4. ForEach控制器->HTTP请求

5. 测试计划->察看结果树

**foreach控制器配置**



注：

* + - 1. 用户定义的变量书写格式(前缀+\_+数字)，例：name\_1
      2. 输入前缀变量：输入要遍历变量前缀
      3. Start index for loop(exclusive):遍历变量开始的索引(从0开始)
      4. End index for loop(inclusive):遍历变量结束的索引(不包括结束索引)
      5. 输出变量名称:定义要被引用的变量名称
    1. 循环控制器

作用：指定循环控制器内取样器的执行次数

例如：执行查询学院接口10次

**使用方法及步骤**

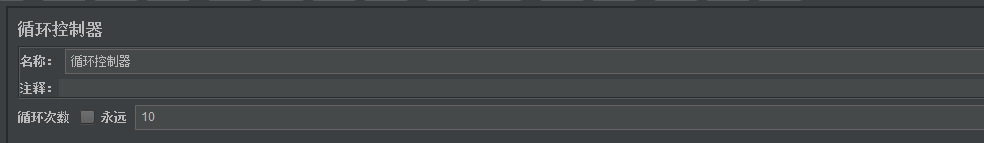
1. 测试计划->线程组

2. 线程组->循环控制器

3. 循环控制器->HTTP请求(查询学院接口)

4. 测试计划->察看结果树

**循环控制器配置**

****

循环次数：要执行的次数

**思考:线程组属性可以控制循环次数.循环控制器有什么作用?**

循环控制器更灵活,循环控制器可以控制线程组中某个请求的执行次数,而线程组属性的循环次数是控制线程组内所有的请求循环次数.

1. jmeter中各元件总结
   1. 组件的作用域
      1. 什么是组件

Jmeter中的每一个功能实现我们就称之为是一个组件,比如我们创建的线程组、http请求、察看结果树...

* + 1. 组件作用域

组件的作用范围

**分类:**

1.取样器是作用域的参考物 ,本身无所谓作用域的概念

2.以结果树为代表的大部分组件

作用域:直接父级取样器 + 直接兄弟取样器

3.以逻辑控制器为代表的组件

只对子级取样器有效.

* 1. 元件执行顺序
     1. 什么是元件

相同类似功能组件的集合称之为元件

* + 1. 元件作用域

jmeter中共有8类可被执行的元件（test plan和thread group不属于元件），其中，sampler（取样器）是不与其他元件发生交互的作用的元件，

Logic Controller（逻辑控制器）只对其子节点的sampler有效，而其他元件需要与sampler等元件交互。

**Config Elements（配置元件）：**影响其范围内的所有元件

**Pre-porcessors（前置处理器）：**在其作用范围内的每一个sampler元件之前执行

**Timer（定时器）：**对其作用范围内的每一个sampler有效

**Post-porcessors（后置处理器）：**在其作用范围内的每一个sampler元件之后执行

**Assirtions（断言）：**对其作用范围内的每一个sampler元件执行后的结果执行校验

**Listener（监听器）：**收集其作用范围内的每一个sampler元件的信息并且呈现出来

在jmeter中，元件的作用域是靠test plan的树形结构中元件的父子关系来确定的，其原则如下：

**1）** sampler不与其他元件相互作用，因此不存在作用域问题

**2）** Logic Controller只对其子节点中的sampler和Logic Controller作用

**3）** 除sampler和Logic Controller外的其他元件，如果是某个sampler的子节点，则该元件仅对其父节点作用

**4）** 除sampler和Logic Controller外的其他元件，如果其父节点不是sampler，则其作用域是该元件父节点下的其他所有后代节点（包括子节点，子节点的子节点等）

* + 1. 元件执行顺序

在同一作用域范围内，test plan中的元件按照以下顺序执行：

1） **Config Element--🡪配置元件**

2） **Pre-porcessors--🡪前置处理器**

3） **Timer-----------🡪定时器**

4） **Sampler---------🡪取样器**

5） **Post-porcessors**（除非Sampler得到的返回结果为空）--🡪后置处理器

6） **Assertions**（除非Sampler得到的返回结果为空）-------🡪断言

7） **Listener**（除非Sampler得到的返回结果为空）--------🡪监听器

注意:Pre-porcessors、Post-porcessors和Assirtions等元件仅对Sampler作用，如在它们作用域内没有任何Sampler，则不会被执行；

    如果在同一作用域范围内有多个同一类型的元件，则这些元件按照它们在test plan中的上下顺序依次执行。

1. jmeter性能测试
   1. 性能测试基础
      1. 什么是软件性能&性能测试

**定义**：软件的性能是软件的一种**非功能特性**，它关注的不是软件是否能够完成特定的功能，而是在完成该功能时展示出来的**及时性**。

由定义可知性能关注的是软件的非功能特性，所以一般来说性能测试介入的时机是在功能测试完成之后。另外，由定义中的及时性可知性能也是一种指标，可以用时间或其它指标来衡量，通常我们会使用某些工具或手段来检测软件的某些指标是否达到了要求，这就是性能测试。

**性能测试定义**：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。

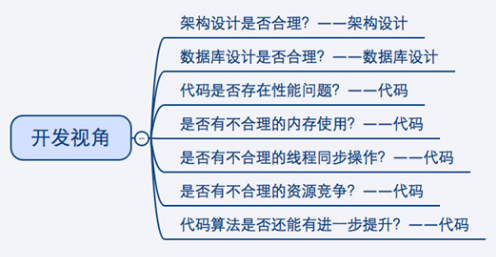
* + 1. 不同人眼中的软件性能

不同的人由于人生观、世界观、价值观以及教育背景、知识体系、人生阅历的不同，对于同一事物或问题的看法可能不同。对于软件性能也是如此，不同的人由于视角的不同，所关注的点也可能不同。下面来看看在不同的人群眼中性能分别是什么样的。

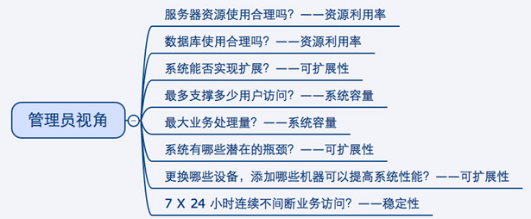
**用户眼中的性能：**



**开发眼中的性能：**



**系统管理者眼中的性能：**



**测试眼中的性能：**

测试人员通常是做为软件质量控制的一个角色，不仅仅是找bug，需要对整个软件的质量负责，性能也属于质量的一部分，因此测试人员眼中的性能应该是全面的，考虑的东西也需要全面：

1、测试人员需要考虑全面的性能，包括用户、开发、管理员等各个视角的性能。

2、测试人员在做性能测试时除开要关注表面的现象如响应时间，也需要关注本质，比如用户看不到的服务器资源利用率，架构设计是否合理？代码是否合理等方方面面。

服务器硬件性能,根据需求和历史数据指定性能目标,建立性能通过模型,对开发代码框架和硬件框架进行性能分析,针对开发发布版本的基准测试,执行软件性能验收及稳定性测试,生产环境的配置和优化,制定性能测试的测试用例,制定性能测试的场景设计,协调各部门配合,特定的性能分析.

* + 1. 性能测试类型和相关术语
       1. 负载:模拟业务操作对服务器造成压力的过程,比如模拟100个用户 发帖.
       2. 基准测试：在给系统施加较低压力时，查看系统的运行状况并记录相关数做为基础参考
       3. 负载测试Load Test：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 。
       4. 压力测试Stress Test：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，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。
       5. 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。
       6. 并发测试：测试多个用户同时访问同一个应用、同一个模块或者数据记录时是否存在死锁或者其他性能问题
    2. 性能测试应用场景（领域）
       1. 性能测试应用场景（领域）主要有：能力验证、规划能力、性能调优、缺陷发现、性能基准比较，下表简单介绍和对比了这几个场景的各自用途和特点：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 主要用途 | 典型场景 | 特点 | 常用性能测试方法 |
| 能力验证 | 关注在给定的软硬件条件下，系统能否具有预期的能力表现 | 在要求平均响应时间小于2秒的前提下，如何判断系统是否能够支持50万用户/天的访问量？ | a)要求在已确定的环境下运行 b)需要根据典型场景设计测试方案和用例，包括操作序列和并发用户量，需要明确的性能目标。 | a)负载测试 b)压力测试 c)稳定性能测试 |
| 规划能力 | 关注如何使系统具有我们要求的性能能力 | 某某系统计划在一年内获客量在到xxx万，系统到时候是否能支持这么多用户量？如果不能需要如何调整系统的配置？ | a) 它是一种探索性的测试 b) 常用于了解系统性能和获得扩展性能的方法 | a) 负载测试 b) 压力测试 c) 配置测试 |
| 性能调优 | 主要用于对系统性能进行调优 | 某某系统上线运行一段时间后响应速度越来越慢，此时应该如何办？ | 每次只改变一个配置，切忌无休止的调优 | a) 并发测试 b) 压力测试 c) 配置测试 |
| 缺陷发现 | 发现缺陷或问题重现、定位手段 | 某些缺陷只有在高负载的情况下才能暴露出来，如线程锁、资源竞争或内存泄露。 | 做为系统测试的补充，用来发现并发问题，或是对系统已经出现的问题进行重现和定位 | a) 并发测试 b) 压力测试 |
| 性能基准比较 | 常用于敏捷开发过程中，敏捷开发流程的特点是小步快走，快速试错，迭代周期短，需求变化频繁。很难定义完善的性能测试目标，也没有时间在每个迭代开展详细的性能测试，可以通过建立性能基线，通过比较每次迭代中的性能表现变化，判断迭代是否达到了目标。 | | | |

* + - 1. 通常在某个性能场景（领域）中需要联合使用多种性能测试方法一起进行性能测试，下表为性能测试应用领域与测试方法关联：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 能力验证 | 规划能力 | 性能调优 | 缺陷发现 | 性能基准比较 |
| 基准测试 | √ |  | √ |  |  |
| 负载测试 | √ | √ | √ |  |  |
| 压力测试 | √ | √ | √ | √ | √ |
| 并发测试 |  |  |  | √ | √ |
| 稳定性测试 | √ |  |  |  |  |

* + 1. 性能测试指标

**性能测试指标概念及目的：**

性能测试是通过测试工具模拟多种正常、峰值及异常负载条件来对系统的各项性能指标进行测试。

目的：验证软件系统是否能够达到用户提出的性能指标，发现系统中存在的性能瓶颈并加以优化。

**性能指标分为两类：**

系统指标（与用户场景和需求相关指标）

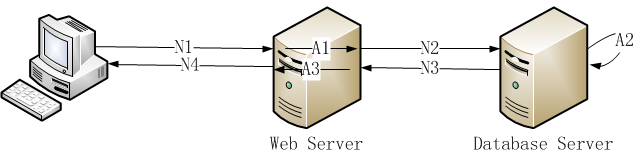
资源指标（与硬件资源消耗相关指标）

**系统指标说明**

**1.响应时间、平均响应时间**

对**一个**请求做出响应所需要的时间

响应时间=网络响应时间+应用程序响应时间=（N1+N2+N3+N4）+(A1+A2+A3)



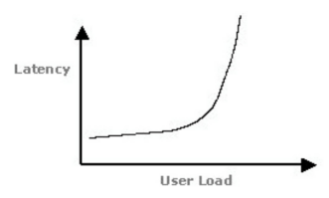
平均响应时间：**所有**请求花费的平均时间

如：如果有100个请求，其中 98 个耗时为 1ms，其他两个为 100ms

平均响应时间： (98 \* 1 + 2 \* 100) / 100.0 = 2.98ms，但是，2.98ms并不能反映服务器的整体效率，因为98个请求耗时才1ms，引申出**百分位数**

百分位数：以响应时间为例，指的是 99% 的请求响应时间，都处在这个值以下，更能体现整体效率。

响应时间和负载的关系



图中拐点说明：

　　　　　　1、响应时间突然增加

　　　　　　2、意味着系统的一种或多种资源利用达到的极限

　　　　　　3、通常可以利用拐点来进行性能测试分析与定位

**2.并发用户数**

并发主要是针对**服务器**而言，在同一时刻与服务器进行交互（指向服务器发出请求）的在线用户数。

**并发用户数**：某一物理时刻同时向系统提交请求的用户数，提交的请求可能是同一个场景或功能，也可以是不同场景或功能。

**在线用户数**：某段时间内访问系统的用户数，这些用户并不一定同时向系统提交请求

**系统用户数**：系统注册的总用户数据

**三者之间的关系：**系统用户数 >= 在线用户数 >= 并发用户数

并发用户数C，计算公式C=nL/T

n：每天访问系统的用户数

L：在线用户从登陆到退出的时间

T：用户每天使用系统大概多长时间

峰值C1,即最大并发数，计算公式C1=C+³√C

本次假设业务量=用户量

1. 业务量：40万\*80%=32万
2. 时间段：24小时\*20%=4.8小时
3. 每小时的业务量：32万/4.8小时=6.67万/小时
4. 每秒的业务量66700/3600=18.53笔/秒 即每个user迭代一次的时间：1/18.53=0.053秒
5. 每人每笔业务的处理时间要求=2秒，2/0.053=37.7个user

小结：系统要达到40万的业务访问量，需要38个并发用户持续运行4.8小时，如果满足则性能测试通过。

**3.吞吐量、吞吐率**

衡量**网络**性能的重要指标

吞吐量：网络传输的数据量（处理客户的请求数）throughput[ˈθru:pʊt]

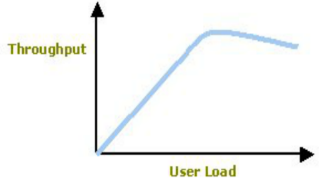
吞吐率：单位时间（可以是秒/分/时/天）内网络成功传输的数据量，如请求数/秒、页面数/秒

从业务角度看，吞吐量可以用：请求数/秒、页面数/秒、人数/天或处理业务数/小时等单位来衡量

从网络角度看，吞吐量可以用：字节/秒来衡量

对于交互式应用来说，吞吐量指标反映的是服务器承受的压力，他能够说明系统的负载能力

吞吐量和负载之间的关系:



图中拐点说明：

　　　　　　1、吞吐量逐渐达到饱和

　　　　　　2、意味着系统的一种或多种资源利用达到的极限

　　　　　　3、通常可以利用拐点来进行性能测试分析与定位

**4.事务，TPS（Transaction Per Second） [trænˈzækʃn]**

**QPS（**Query Per Seconds**）**

事务：可以看作是一个动作或是一系列动作的集合，例如登录，从登录开始到登录结束为一个事务。

TPS：衡量**系统**处理事务或交易的能力，每秒处理的事务数

QPS：服务器在一秒的时间内处理了多少个请求，也即是最大吞吐能力。

**QPS（TPS）= 并发数/平均响应时间**

 一个系统吞吐量通常由QPS（TPS）、并发数两个因素决定，每套系统这两个值都有一个相对极限值，在应用场景访问压力下，只要某一项达到系统最高值，系统的吞吐量就上不去了，如果压力继续增大，系统的吞吐量反而会下降，原因是系统超负荷工作，上下文切换、内存等等其它消耗导致系统性能下降。

**5.点击量、点击率（Hits Per Second）**

点击数：指Web Server收到的HTTP请求数。

点击率：**单位时间每秒**用户向Web Server提交的HTTP请求数。

区分鼠标点击数：如请求一个网页，网页含有3张图片，向Web Server请求的点击数：1+3=4，而鼠标的一次点击就可以访问网页，点击数只有1次

**6.PV和UV**

PV：访问一个URL，产生一个PV（Page View，页面访问量），每日每个网站的总PV量是形容一个 网站规模的重要指标。

UV：作为一个独立的用户，访问站点的所有页面均算作一个UV（Unique Visitor，用户访问）

单台服务器每天PV计算  
公式1：每天总PV = QPS \* 3600 \* 6

公式2：每天总PV = QPS \* 3600 \* 8

**资源指标说明**

**1.硬件性能指标—资源占用率**：指的是对不同系统资源的使用程度，通常以占用最大值的百分比来衡量**Utilization [ˌju:təlaɪ'zeɪʃn]**

**通常需要关注的服务器资源如下：**

CPU，内存Memory，磁盘I/O（Disk I/O），网络I/O（Network I/O）

**CPU**：主要解释计算机指令以及处理计算机软件中的数据

Linux系统中top命令查看CPU的使用率

CPU的利用率（<=75%）有：user（用户使用）,sys（系统调用<=30%）,wait（等待<=5%），idle（空闲）

当user消耗高时，通过top命令查看哪个用户进程占用cpu的使用

user消耗过高的原因可能有:

（1）代码问题。如代码中耗时循环中不加sleep，即例如while的死循环中，没有加sleep时间，导致没有空余的时间将cpu的控制权给其他的进程，一直陷入该死循环中，cpu得不到休息，所以user的消耗过高，则cpu的消耗高

（2）gc频繁。gc则为垃圾回收，由于垃圾回收也是需要大量的计算，也消耗cpu，所以当gc频繁时也导致usr用户空间的消耗也过高，cpu消耗过高

当sys消耗高时，通过top命令查看系统调用资源的情况

sys消耗过高的原因可能有：  
（1）上下文切换频繁。上下文切换发生的情况有：中断处理，多任务处理，用户状态改变。中断处理，当cpu停止处理当前的进程转而处理中断请求的进程时发生上下文切换。多任务处理则为有多个进程请求cpu的处理，进程的数量多于cpu的核数，则分配进程时间片，根据时间片处理进程，意味着会强制停止一个进程而去处理另一个进程，形成频繁的上下文切换。用户状态改变则为user状态与sys状态的改变。

wait较高时，即等待的进程占比高则可以考虑是否磁盘读写，磁盘瓶颈问题， 等待的进程较多时，cpu无论如何切换都是切换到等待的进程，导致cpu一直在频繁切换等待的线程而利用率较低

**内存**：与cpu沟通的桥梁，计算机中所有程序的运行都在内存中进行，内存分为物理内存、页面交换（Paging），SWAP内存（虚拟内存）

页面交换：当物理内存即实际的内存满了的时候，将物理内存中不常用的进程调出存储到虚拟内存中，以缓解物理内存空间的压力，所以当物理内存与虚拟内存的数据交换频繁的时候，这时候就要关注下内存的性能情况。

SWAP内存：为进程分配虚拟的内存空间，即调用硬盘的空间作为内存使用。

内存的性能分析是由可用内存与页面交换来分析的，可用内存使用占70%-80%为上限，当超出这个数值，内存性能情况就比较危险，而且即使可用内存使用不超过80%的数值时，页面交换比较频繁时，也是要关注下内存情况

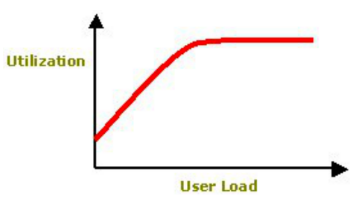
一般物理内存即使是满内存也不能代表内存出现问题，主要是看虚拟内存swap，虚拟内存应该<=70%，大于则可以考虑是否内存问题或者内存泄漏

**磁盘吞吐量**，指单位时间内通过磁盘的数据量。主要关注磁盘的繁忙率，如果高于70%，则磁盘瓶颈

**网络吞吐量**，指单位时间内通过网络的数据量，当吞吐量大于网路设备或链路最大传输能力，即带宽时，则应该考虑升级网络设备或者增加带宽，Linux命令netstate

网络IO也有可能出现终止连接失败。例如当服务端出现大量的TIME\_WAIT，TCP终止连接的第4个步骤，在主动发起关闭连接方接收到结束符FIN时状态变为TIME\_WAIT,这时在服务端发现大量的TIME\_WAIT,意味着关闭连接是由服务端发起的。

资源利用率和负载的关系:



图中拐点说明：

　　　　　　1、服务器某个资源使用逐渐达到饱和

　　　　　　2、通常可以利用拐点来进行性能测试分析与定位

**2.中间件：**常用的中间件例如web服务器Tomcat,Weblogic web服务器，JVM（java虚拟机），ThreadPool线程池，JDBC数据驱动

注：http服务器和web服务器与应用服务器的差别是一个存储静态网页的服务器，一个是存储css，js等动态加载网页的服务器，而tomcat则属于应用服务器

注：对中间件例如对服务器的性能测试，需要将监控的jmeter-server的文件下载在服务端上，然后开启即可监控被测服务器的性能，或者将监控的软件下载在被测服务器上，远程启动监控软件等

**3.数据库指标**

应关注SQL，吞吐量，缓存命中率，连接数等，则是关注sql语句执行时间，以微妙为单位，吞吐量TPS，缓存命中率应>=95%

注：对数据库的性能测试通过jemter利用批量的sql语句对数据库进行操作，从而测试数据库的性能，前提是需要将jdbc的驱动加载在测试计划添加的驱动文件中，然后添加jdbc的前置处理器和jdbc的请求sample。

**4.JVM，**java虚拟机，为使java的代码可以编译运行在不同的平台上顺畅，仿真模拟各种计算机来实现

GC：自动内存管理程序，被引用的对象保存在内存中，当对象不被引用时则释放。关注的参数有Full GC完全java虚拟机垃圾部分回收频率

**5.前端指标**

前端应该关注页面展示，即首次显示时间，页面数量，页面大小，网络startRender，firstRender等

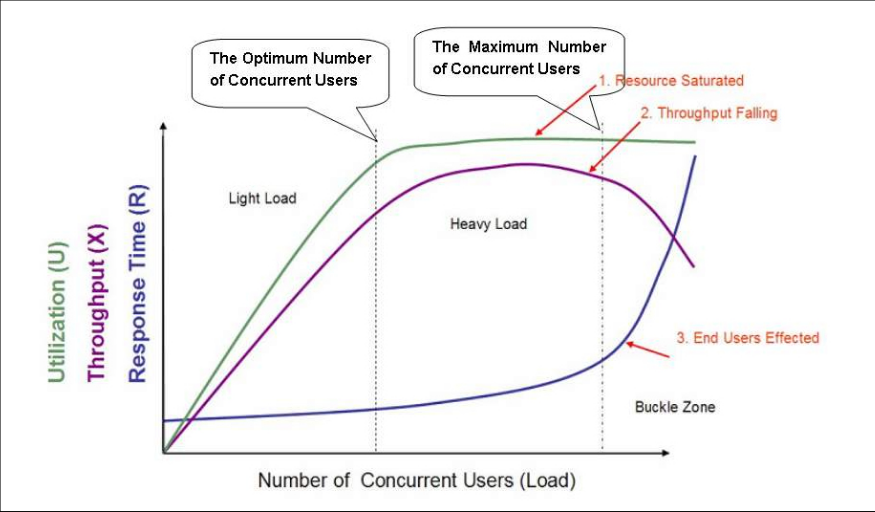
注：关注前端的性能与后端的性能的不同点在于，前端是每个用户的直观的感受，以及前端页面的加载元素耗费时间给予用户的感受，而后端的性能关注点在于多用户使用系统时，服务器是否能够承受或者服务器的处理能力如何，能否以较好的响应时间响应。

**6.Load：**系统平均负载，特定时间间隔内运行进程数，Load与cpu核数一致

**注：理解最佳并发用户数和最大并发用户数**

性能测试——理发店模型

所谓的理发店模式，简单地阐述一下，一个理发店有3个理发师，当同时来理发店的客户有3个的时候，那么理发师的资源能够有效地利用，这时3个用户数即为最佳的并发用户数；当理发店来了9个客户的时候，3个客户理发，而6个用户在等待，3个客户的等待时间为1个小时，另外的3个客户的等待时间为2小时，客户的最大忍受时间为3小时包括理发的1个小时，所以6个客户的等待时间都在客户的可以承受范围内，故9个客户是该理发店的最大并发用户数。具体的随着并发用户数的增加，响应时间，吞吐量，资源利用情况如下图所示：



这张图中展示的是1个标准的软件性能模型。在图中有三条曲线，分别表示资源的利用情况（Utilization，包括硬件资源和软件资源）、吞吐量（Throughput，这里是指每秒事务数）以及响应时间（Response Time）。图中坐标轴的横轴从左到右表现了并发用户数（Number of Concurrent Users）的不断增长。

在这张图中我们可以看到，最开始，随着并发用户数的增长，资源占用率和吞吐量会相应的增长，但是响应时间的变化不大；不过当并发用户数增长到一定程度后，资源占用达到饱和，吞吐量增长明显放缓甚至停止增长，而响应时间却进一步延长。如果并发用户数继续增长，你会发现软硬件资源占用继续维持在饱和状态，但是吞吐量开始下降，响应时间明显的超出了用户可接受的范围，并且最终导致用户放弃了这次请求甚至离开。

根据这种性能表现，图中划分了三个区域，分别是Light Load（较轻的压力）、Heavy Load（较重的压力）和Buckle Zone（用户无法忍受并放弃请求）。在Light Load和Heavy Load 两个区域交界处的并发用户数，我们称为“最佳并发用户数（The Optimum Number of Concurrent Users）”，而Heavy Load和Buckle Zone两个区域交界处的并发用户数则称为“最大并发用户数（The Maximum Number of Concurrent Users）”。

当系统的负载等于最佳并发用户数时，系统的整体效率最高，没有资源被浪费，用户也不需要等待；当系统负载处于最佳并发用户数和最大并发用户数之间时，系统可以继续工作，但是用户的等待时间延长，满意度开始降低，并且如果负载一直持续，将最终会导致有些用户无法忍受而放弃；而当系统负载大于最大并发用户数时，将注定会导致某些用户无法忍受超长的响应时间而放弃。

**最佳并发用户数**：当系统的负载等于最佳并发用户数时，系统的整体效率最高，没有资源被浪费，用户也不需要等待  
**最大并发用户数**：系统的负载一直持续，有些用户在处理而有的用户在自己最大的等待时间内等待的时候

我们需要保证的是：

（1）最佳并发用户数需大于系统的平均负载

（2）系统的最大并发用户数要大于系统需要承受的峰值负载

怎么理解这两句话呢？

（1）系统的平均负载：在特定的时间内，系统正在处理的用户数和等待处理的用户数的总和

如果系统的平均负载大于最佳并发用户数，则用户的满意度会下降，所以我们需要保证系统的平均负载小于或者等于最佳并发用户数

（2）峰值：指的是系统的最大能承受的用户数的极值

只有最大并发用户数的大于系统所能承受的峰值负载，才不会造成等待空间资源的浪费，导致系统的效率低下

* + 1. 做好性能测试需要掌握的知识
       1. 掌握一门编程语言
       2. 掌握计算机原理和操作系统知识
       3. 良好的网络基础
       4. 掌握数据库知识
       5. 中间件（apache,tomcat）
       6. 常用抓包工具
       7. 性能测试工具
  1. 性能测试流程
     1. 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，后面的性能测试执行其实是没有任何意义的，而且性能需求分析做的好不好直接影响到性能测试的结果。

一些性能测试人员常犯的错误就是测试一开始就直接用工具对系统进行加压，没有弄清楚性能测试的目的，稀里糊涂做完了以后也不知道结果是否满足性能需求。市面上的书籍也大都是直接讲性能测试工具如LR，jmeter如何使用，导致很多新手一提到性能测试就直接拿工具来进行录制回放，使得很多人认为会使用性能测试工具就等于会性能测试了，殊不知工具其实只是性能测试过程中很小的一部分。

在需求分析阶段，测试人员需要与项目相关的人员进行沟通，收集各种项目资料，对系统进行分析，建立性能测试数据模型，并将其转化为可衡量的具体性能指标，确认测试的目标。所以性能测试需求分析过程是繁杂的，需要测试人员有深厚的性能理论知识，除此之外还需要懂一些数学建模的知识来帮助我们建立性能测试模型。

**首先**，让我们来看看通过性能需求分析我们需要得出哪些结论或目标：

明确倒底要不要做性能测试？性能测试的目的是什么？

明确被测系统是什么？被测试系统的相关技术信息如：架构、平台、协议等

明确被测系统的基本业务、关键业务，用户行为

明确性能测试点是什么？哪些需要测，为什么？哪些不需要测，又是为什么？

明确被测系统未来的业务拓展规划以及性能需求？

明确性能测试策略，即应该怎么测试？

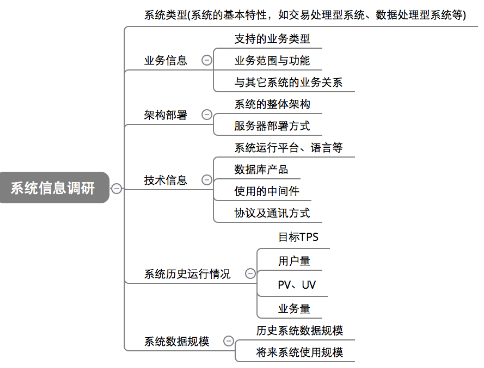
明确性能测试的指标，知道测试出来的结果怎么算通过？

**其次**，需求分析阶段我们可以从以下几个方面入手：

* + - 1. **系统信息调研：**

指对被测试系统进行分析，需要对其有全面的了解和认识，这是我们做好性能测试的前提，而且在后续进行性能分析和调优时将会大有用处，试想如果连系统的架构、协议都不了解，我们如何进行准确的性能测试？如果进行性能分析与调优？

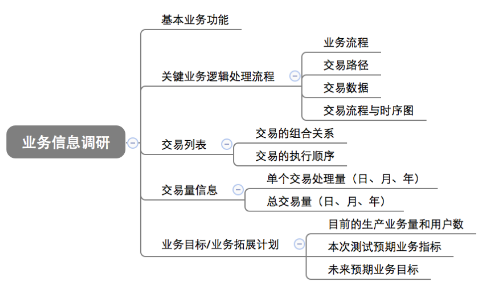
需要分析的系统信息如下（包括但不仅限于如下这些）：



* + - 1. **业务信息调研：**

指对被测试的业务进行分析，通过对业务的分析和了解，方便我们后续进行性能测试场景的确定以及性能测试指标的确定。

需要分析的业务信息如下（包括但不仅限于如下这些）：



* + - 1. **性能需求评估：**

在实施性能测试之前，我们需要对被测系统做相应的评估，主要目的是明确是否需要做性能测试。如果确定需要做性能测试，需要进一步确立性能测试点和指标，明确该测什么、性能指标是多少，测试通过or不通过的标准？性能指标也会根据情况评估，要求被测系统能满足将来一定时间段的业务压力。

判断是否进行性能测试主要从下面两个方面进行思考：

**业务角度:**

系统是公司内部 or 对外？系统使用的人数的多少？如果一个系统上线后基本没几个人使用，无论系统多大，设计多么复杂，并发性的性能测试都是没必要的，前期可以否决。当然，除非在功能测试阶段发现非常明显的性能问题，使得用户体验较差的，此时可进行性能测试来排查问题。

**系统角度：**系统又可以从以下3个方面进行分析

a）系统架构：

如果一个系统采用的框架是老的系统框架（通常大公司都有自己的统一框架），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，因为老框架的使用肯定是经过了验证的。如果一个系统采用的是一种新的框架，可以考虑做性能测试。

b）数据库要求：

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。

c）系统特殊要求：

从实时性角度来分析，某些系统对响应时间要求比较快，比如证券系统，系统的快慢直接影响客户的收益，这种情况就有作并发测试的必要，在大并发量的场景下，查看这个功能的响应时间。

从大数据量上传下载角度分析，某些系统经常需要进行较大数据量的上传和下载操作，虽然此种操作使用的人数不会太多，但是也有必要进行性能测试，确定系统能处理的最大容量，如果超过这个容量时系统需要进行相关控制，避免由于人工误操作导致系统内存溢出或崩溃。

* + - 1. **确定性能测试点：**

在上面第3点中，我们简单分析了如何确定一个系统是否需要做性能测试。下面简单总结下如果一个系统确定要做性能测试，我们如何确定被测系统的性能测试点？

我们可以从下面几个方面进行分析：

**关键业务：**

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。例如转账，扣款等接口。如果项目（或功能点）不属于关键业务（或关键业务点），则可转入下面。

**日请求量：**

确定被测项目各功能点的日请求量（可以统计不同时间粒度下的请求量如：小时，日，周，月）。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，而且关键业务点，可以被确定为性能点。

**逻辑复杂度：**

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。

**运营推广活动：**

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。

以上 4 点，是相辅相成、环环相扣的。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 点，但又属于资源高消耗（内存、CPU），也可列入性能测试点行列。

* + - 1. **确定性能指标：**

性能指标的确定是根据上述4点所提炼出来的。

* + 1. 测试准备
       1. **测试环境准备：**

a）系统运行环境：这个通常就是我们的测试环境，有些时候需求比较多，做性能测试担心把环境搞跨了影响其它的功能测试，可能需要重新搭建一套专门用来做性能测试的环境。

b）执行机环境：这个就是用来生成负载的执行机，通常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准备好执行机环境。

* + - 1. **测试场景设计：**根据性能需求分析来设计符合用户使用习惯的场景，场景设计的好不好直接影响到性能测试的效果。
      2. **性能工具准备：**

a）负载工具：根据需求分析和系统特点选择合适的负载工具，比如LR、Jmeter等

b）监控工具：准备性能测试时的服务器资源、JVM、数据库监控工具，以便进行后续的性能测试分析与调优。

* + - 1. **测试脚本准备：**如果性能测试工具不能满足被测系统的要求或只能满足部分要求时，需要我们自己开发脚本配合工具进行性能测试。
      2. **测试数据准备：**

a）负载测试数据：并发测试时需要多少数据？比如登录场景？

b）DB数据量大小：为了尽量符合生产场景，需要模拟线上大量数据情况，那么要往数据库里提前插入一定的数据量。这可能需要花费一些时间，特点是关联系统较多，逻辑复杂的业务可能同时涉及多张表。

* + - 1. **其它：**如果需要其它关联系统或专业人士如DBA配合的，也需要提前进行沟通。
    1. 测试执行
       1. 人工边执行边分析

通常我们做性能测试都是人工执行并随时观察系统运行的情况、资源的使用率等指标。性能测试的吸引力之一就在于它的不可预知性。当我们在做性能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。但这个过程可能会很慢长，需要不断的调整系统配置或程序代码来定位问题，耗时耗人力。特别是在当前敏捷开发模式比较流行的大环境下，版本发布非常频繁且版本周期短（通常1~2周一个版本），没有那么长的时间来做性能测试。

* + - 1. 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执行过程分离，执行过程只是机器服从指令的运行而已。通常测试环境在白天比较繁忙，出现性能问题及定位难度较大且会影响功能测试。所以一般性能测试最好在晚上或周末进行，在相对较安静的条件有利于测试结果的稳定性。这种方法也相对比较适合敏捷的模式，不需要人工一直守着。只需要在拿到结果后进行分析就好了。同进，这种方式对测试人员能力的要求比较高，需要我们能进行自动化的收集各种监控数据、生成报表便于后续分析。

* + 1. 结果分析与调优（不单单是测试的事情）

通过对性能测试结果分析找出系统瓶颈；通过调优优化系统消除系统瓶颈。

目的：使系统运行的更好

* + - 1. **一般系统的瓶颈**

 性能测试调优需要先发现瓶颈，那么系统一般会存在哪些瓶颈：

**硬件上的性能瓶颈**：

一般指的是CPU、内存、磁盘I/O 方面的问题，分为服务器硬件瓶颈、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、数据库、web服务器等）、应用瓶颈（SQL 语句、数据库设计、业务逻辑、算法等）。

**例如：确定了在服务器数据库上需要6个CPU、12GB内存，但是在测试时发现，CPU的持续利用率超过95%，这时可以认为在硬件上出现了性能瓶颈。**

**应用软件上的性能瓶颈：**

一般指的是应用服务器、web 服务器等应用软件，还包括数据库系统。

例如：在weblogic平台上配置了JDBC连接池的参数，最大连接数为50，最小连接数为5，增加量为10。测试时发现，当负载增加时，现有的连接数不足，系统会动态生成10个新的连接，导致交易处理的响应时间大大增加。这时可以认为在应用软件上出现了性能瓶颈。

**应用程序上的性能瓶颈：**

一般指的是开发人员新开发出来的应用程序。

**例如：程序员开发了一个缴费处理程序，在测试时发现，这个缴费处理程序在处理用户并发缴费请求时，只能串行处理，无法并行处理，导致缴费处理响应时间非常长，这个可以认为在应用程序上出现了性能瓶颈。**

**操作系统上的性能瓶颈：**

一般指的是windows、UNIX、Linux等操作系统。

例如，在进行性能测试，出现物理内存不足时，虚拟内存设置也不合理，虚拟内存的交换效率就会大大降低，从而导致行为的响应时间大大增加，这时认为操作系统上出现性能瓶颈。

**网络设备上的性能瓶颈**：

一般指的是防火墙、动态负载均衡器、交换机等设备。

例如，在动态负载均衡器上设置了动态分发负载的机制，当发现某个应用服务器上的硬件资源已经到达极限时，动态负载均衡器将后续的交易请求发送到其他负载较轻的应用服务器上。在测试时发现，动态负载均衡器没有起到相应的作用，这时可以认为网络瓶颈。

性能测试出现的原因及其定位十分复杂，这里只是简单介绍常见的几种瓶颈类型和特征，而性能测试所需要做的就是根据各种情况因素综合考虑，然后协助开发人员\DBA\运维人员一起定位性能瓶颈。

**查找瓶颈时按以下顺序，由易到难。**

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，数据库，web服务器等）---〉应用瓶颈（SQL语句、数据库设计、业务逻辑、算法等）

注：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。对一些要求低的，我们分析到应用系统在将来大的负载压力（并发用户数、数据量）下，系统的硬件瓶颈在哪儿就够了。  
     **分段排除法 很有效**

* + - 1. **一般性能调优步骤**

 一般性能问题调优的步骤：

**步骤一：**确定问题

应用程序代码：在通常情况下，很多程序的性能问题都是写出来的，因此对于发现瓶颈的模块，应该首先检查一下代码。

数据库配置：经常引起整个系统运行缓慢，一些诸如oracle 的大型数据库都是需要DBA进行正确的参数调整才能投产的。

操作系统配置：不合理就可能引起系统瓶颈。

硬件设置：硬盘速度、内存大小等都是容易引起瓶颈的原因，因此这些都是分析的重点。

网络：网络负载过重导致网络冲突和网络延迟。

**步骤二：**确定原因

当确定了问题之后，我们要明确这个问题影响的是响应时间吞吐量，还是其他问题？是多数用户还是少数用户遇到了问题？如果是少数用户，这几个用户与其它用户的操作有什么不用？系统资源监控的结果是否正常？CPU的使用是否到达极限？I/O 情况如何？问题是否集中在某一类模块中？ 是客户端还是服务器出现问题？ 系统硬件配置是否够用？实际负载是否超过了系统的负载能力？ 是否未对系统进行优化？

通过这些分析及一些与系统相关的问题，可以对系统瓶颈有更深入的了解，进而分析出真正的原因。

**步骤三：** 确定调整目标和解决方案

提高系统吞吐量，缩短响应时间，更好地支持并发。

**步骤四：**测试解决方案

对通过解决方案调优后的系统进行基准测试。（基准测试是指通过设计科学的测试方法、测试工具和测试系统，实现对一类测试对象的某项性能指标进行定量的和可对比的测试）

**步骤五：**分析调优结果

系统调优是否达到或者超出了预定目标？系统是整体性能得到了改善，还是以系统某部分性能来解决其他问题。调优是否可以结束了。

最后，如果达到了预期目标，调优工作就基本可以结束了。

**性能调优要点：**

要点1： 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。

要点2： 确定清晰明确的性能目标是关键。

要点3： 必须保证调优后的程序运行正确。

要点4： 系统的性能更大程度上取决于良好的设计，调优技巧只是一个辅助手段。

要点5： 调优过程是迭代渐进的过程，每一次调优的结果都要反馈到后续的代码开发中去。

要点6： 性能调优不能以牺牲代码的可读性和可维护性为代价。

* + 1. 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。

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。

* + 1. 常见性能问题及成因

**常见性能问题的六个特征**

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。

**常见性能问题成因**

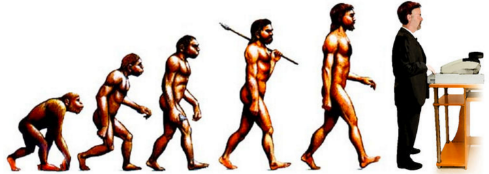
常见性能问题及成因列表：

|  |  |  |  |
| --- | --- | --- | --- |
| 性能问题 | 描述 | 问题特性 | 成因 |
| 线性内存泄露 | 每单位(每事务、每用户等)泄漏造成内存随着时间或负载线性增长。这会随着时间或负载增长降低系统性能。只有重启才有可能恢复。 | 随着时间越来越慢 随着负载越来越慢 | 虽然可能有多种外部原因，但最典型的是与资源泄漏有关(例如，没有清理不能被GC回收的大对象，引起大对象不断积累)。 |
| 指数方式内存泄露 | 双倍增长的内存泄漏造成系统内存消耗表现为时间的指数曲线 | 随着时间越来越慢 随着负载越来越慢 | 通常是由于向集合(Vector，HashMap) 中加入永远不删除的元素造成的。 |
| 糟糕的编码：无限循环 | 线程在 while(true) 语句以及类似的语句里阻塞。 | 可以预见的锁定 | 需要对循环进行大刀阔斧的删剪。 |
| 资源泄漏 | JDBC 语句，CICS 事务网关连接，以及类似的东西被泄漏了，造成桥接层和后端系统的影响。 | 随着时间越来越慢 可以预见的锁定 突然混乱 | 通常情况下，这是由于遗漏了 finally 块，或者更简单点，就是忘记用 close() 关闭外部资源的对象所造成的。 |
| 外部瓶颈问题 | 后端或者其他外部系统（如鉴权）越来越慢，同样减缓了应用服务器和应用程序 | 持续缓慢 随着负载越来越慢 | 咨询专家（负责的第三方或者系统管理员），获取解决外部瓶颈问题的方法。 |
| 外部系统 | 应用程序通过太大或太多的请求滥用后端系统。 | 持续缓慢 随着负载越来越慢 | 清除冗余的工作请求 ，批量处理相似的工作请求，把大的请求分解成若干个更小的请求，调整工作请求或后端系统(例如，公共查询关键字的索引)等。 |
| 糟糕的编码：CPU密集的组件 | 一些糟糕的代码进行交互处理时，就挂起了 CPU，把吞吐速度减慢到爬行的速度。 | 持续缓慢 随着负载越来越慢 | 典型的解决方案就是数据高速缓存。 |
| 中间层问题 | 实现得很糟糕的桥接层(如JDBC 驱动程序、数据库连接池管理)，由于对数据和请求不断的排列、解除排列，从而把所有通过它的流量减慢到爬行速度。这个毛病在早期阶段很容易与外部瓶颈混淆。 | 持续缓慢 随着负载越来越慢 | 检查桥接层和外部系统的版本兼容性。如果有可能，评估不同的桥接供应商。如果重新规划架构，有可能完全不需要桥接。 |
| 内部资源瓶颈：过度使用或分配不足 | 内部资源(线程池、对象池)变得稀缺。是在正确使用的情况下加大负载时出现过度使用还是因为泄漏？ | 随着负载越来越慢 零星的挂起或异常错误 | 分配不足：根据预期的最大负载提高池的最大尺寸。过度使用：请参阅外部系统的过度使用。 |
| 不停止的重试 | 这包括对失败请求连续的(或者在极端情况下无休止的)重试。 | 可以预见的锁定 突然混乱 | 可能是后端系统完全当机，或者网络连接中断。 |
| 线程：阻塞点 | 线程遇到同步阻塞，造成交通阻塞。 | 随着负载越来越慢 零星的挂起或异常错误 可以预见的锁定 | 可能同步是不必要的(只要重新设计)，或者比较外在的锁定策略(例如，读/写锁)也许会有帮助。 |
| 线程：死锁/活动锁 | 这是基本的“获得顺序”的问题。 | 突然混乱 | 获得顺序”的算法不合理。 |

* 1. 性能测试工具选型

性能测试和功能测试不同，性能测试的执行是基本功能的重复和并发，需要模拟多用户，在性能测试执行时需要监控指标参数，同时性能测试的结果不是那么显而易见，需要对数据进行分析。这些特点决定了性能测试更适合通过工具来完成。

* + 1. 为什么要使用工具
       1. 从人类进化的角度来看，会制造并使用工具是人和猿人最根本的区别，因为工具可以帮助我们提高生产力和效率。



* + - 1. 想象下如果不使用工具进行性能测试会怎么样？

我们可以从**性能测试的定义**的角度来分析**，**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。如果不使用工具，仅靠人工进行性能测试会存在以下的弊端：

**a）测试需要投入大量的资源**：

为了模拟多种负载、并发的场景需要多人协同工作，通常测试没有很多的资源，而且就算有资源人工的效果也会大打折扣，甚至于某些场景仅凭人工是无法完成的。

**b）可重复性非常差**：

性能测试经常需要反复调优和测试执行，如果没有工具的帮助，全靠人工实在不敢想象。

**c）测试准确性较差**：

由于需要模拟多种负载和并发场景，如果由人工来操作，难免会存在误差，而且相对工具或程序来说这种误差会更大，对测试结果影响也非常大。

**d）结果的收集、整理和呈现形式差**：

如果没有工具，全凭人工采集数据相对工具来说也会存在较大的误差。

* + 1. 性能测试与性能测试工具的关系

1、性能测试从测试阶段来划分属于系统测试，其和具体使用什么工具并没有直接的关系。使用工具只是为了提高性能测试效率和准确性的一种方法和手段。从本质上来看，同做其它事情时使用工具没有什么实质性的区别。

2、性能测试不等于Loadrunner、jmeter，他们只是性能测试工具其中的一种，而且它也不是万能的，在某些情况下它也并不能派上用场。推荐看下《[让LoadRunner走下神坛](http://www.ltesting.net/ceshi/ceshijishu/rjcsgj/mercury/loadrunner/2008/0203/142050.html)》和《[让LoadRunner再次走下神坛](http://wenku.baidu.com/link?url=27pI7xH2iuu0CsDWD1A8VximavdrmpUUJvsdG64PG0r-bXcQ-8GzeQ8I3dfz-puNvLfiISYQCSefPLyfE-VqmS50mQClpTuarxzsNsu2SXS)》这两篇文章于对性能测试和LR的关系讲的比较深刻。

3、自动化测试工具与性能测试工具的区别：性能测试工具一般是基于通信协议的（客户端与服务器交换信息所遵守的约定），它可以不关心系统的UI，而自动化使用的是对象识别技术，关注UI界面。自动化无法或很难造成负载，但是通过协议很容易。

* + 1. 性能测试工具选型参考

通常在公司或项目中，我们选择任何工具时都会做一些调研，目的就是为了选择适合公司或项目的工具。那么性能测试工具也不例外，通常可以从以下几个方面进行考虑：

**1、成本方面**：

**a）工具成本**：工具通常分为商业闭(shou)源(fei)和非商业开(mian)源(fei)两种，商业工具通常功能比较强大，收费，由于收费所以可提供售后服务，如果出了问题有专业人士帮忙处理。而开源工具通常是免费的，功能有限，维护工具的组织也是自发的，所以如果碰到问题需要自行解决，没有专人提供服务。具体选择商业还是开源的工具，需要根据公司的情况，比如公司规模、愿意承担的成本、项目综合情况等方面考虑。一般来看大公司通常可以承担的起工具的费用，会考虑购买商业工具。而小公司由于资金压力，可能会选择开源的工具。

**b）学习成本**：使用任何工具都需要进行学习，这样一来就会产生学习成本（比如：时间），因此我们在选择工具时也需要考虑到项目组成员的学习成本。如果有两种工具A和B都能满足项目组测试的需求，如果A工具大部分人都会使用，而B工具只有极少部分人会使用，那么建议优先考虑A工具。通常，对于测试人员最好熟悉一款流行的商业（性能）工具，一款开源免费（性能）工具，还需要熟悉常见的（性能）脚本开发语言等，这是基本要求。

**2、支持的协议**：

性能测试通常跟协议联系非常紧密，比如B/S的系统通常使用http协议进行客户端和服务器商的信息交换，C/S的系统通常使用socket协议进行信息交换。在选择工具时，需要考虑项目使用的协议。一个测试工具能否满足测试需求，能否达到令人满意的测试结果，是选择测试工具要考虑的最基本的问题。

**3、生命力**：

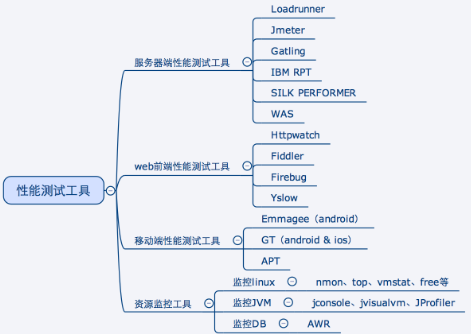
现在的性能测试工具非常多，比如LR，jmeter这类较大众的工具网上相关的资料非常多，但一些小众工具可能网上资料比较少。如果在工具使用过程中碰到了比较棘手的问题，在寻求解决方案或帮助时，大众的的工具相对来说会比较有优势一点，毕竟使用的人越多，资料越多，那么自己碰到的问题也许别人早就碰到并解决了，即时之前没有人碰到过，由于使用研究的人多，通过社区或论坛的帮助相信总会有高手能协助解决的。

**4、跨平台**：

这一点自不必多说，看看JAVA、Python为什么一直这么流行就知道了。

* + 1. 常见性能测试工具

性能测试工具，从理论上来讲在性能测试过程中使用到的所有工具都可以称其为性能测试工具，通常分为以下几类：



说明：

服务器端性能测试工具：需要支持产生压力和负载，录制和生成脚本，设置和部署场景，产生并发用户和向系统施加持续的压力。

web前端性能测试工具：需要关于心浏览器等客户端工具对具体需要展现的页面的处理过程。

移动端性能测试工具：同web端性能测试工具也需要关心页面的处理过程，另外还要具体数据采集的功能，比如：手机CPU、内存、电量，启动时间等数据的记录。

资源监控工具：这个主要是能够收集性能测试过程中的数据以及良好的结果展现方式。

* + 1. 常见性能测试工具的特点

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，而且由于是JAVA编写的，本身比较消耗资源。

LoadRunner：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。

下表为 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，比较灵活 | LoadRunner安装包比较大，安装比较麻烦，工具本身相对比较笨重 |
| 支持的协议 | 支持多种协议：HTTP、HTTPS、SOAP、FTP、Database via JDBC、JMS等，但相对LR还是不够全面，由于此原因相对来说jemter比较灵活，轻便 | 支持的协议非常多，比较全面，但正因此显得工具本身比较笨重，不够灵活 |
| 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 |
| 并发模式 | 通过增加线程组的数目并设置集合点来增加和实现并发 | 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 |

* 1. jmeter性能测试演示
     1. 固定吞吐量测试

**需求说明：**

一个用户以20qps的频率来访问服务器,持续10秒钟.查看服务器平均响应时间

**使用元件：**

线程组——定时器——constant througthput timer（**固定吞吐量定时器**）

实现步骤：

1.创建测试计划

2.创建线程组 --设置循环次数= 访问频率\*持续时间

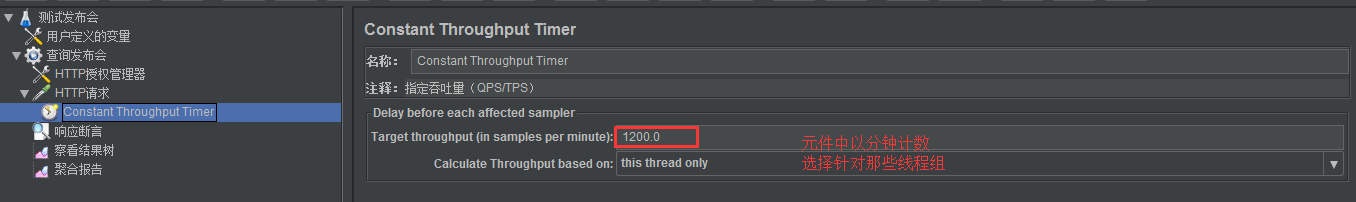
3.添加http请求

4.添加constant througthput timer(常量吞吐定时器)--->设置访问频率

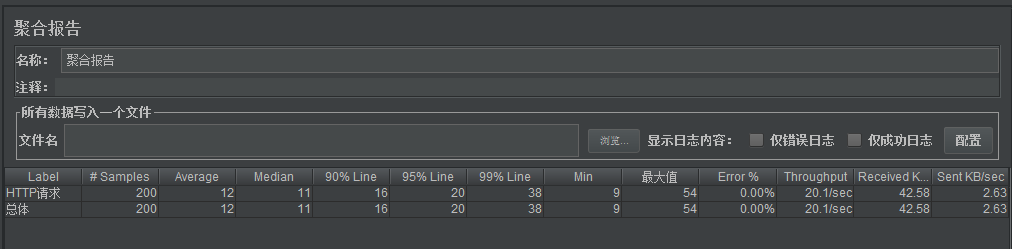
设置访问频率 = qps ×60(因为是以分钟计)

5.添加聚合报告

**constant througthput timer元件配置：**



jmeter----聚合报告参数详解



Label----每个请求的名称，比如HTTP请求等

#Samples----发给服务器的请求数量

Average----单个请求的平均响应时间

Median----50%请求的响应时间

90%Line----90%请求响应时间

95%Line----95%请求响应时间

99%Line----99%请求的响应时间

Min----最小的响应时间

Max----最大的响应时间

Error%----错误率=错误的请求的数量/请求的总数

Throughput----吞吐量即表示每秒完成的请求数

Received KB/sec----每秒从服务器端接收到的数据量

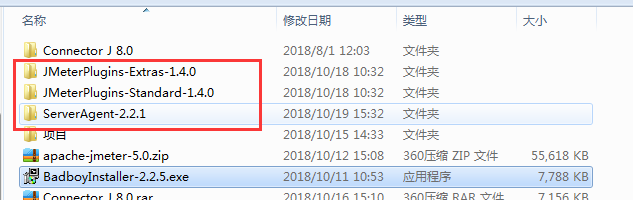
Sent KB/sec----每秒从客户端发送的请求的数量

* + 1. 指定集合点测试（前文已讲过）
    2. jmeter监控服务器性能

固定吞吐量和指定集合点，都是使用jmeter进行性能测试的常用手段，但是通过对测试过程的监控和测试结果的分析，才是在性能测试中值得关注部分之一。

jmeter也可以监控服务器的性能参数，如CPU、内存、IO等，不过需要下载一些插件

* + - 1. 下载需要的jmeter插件



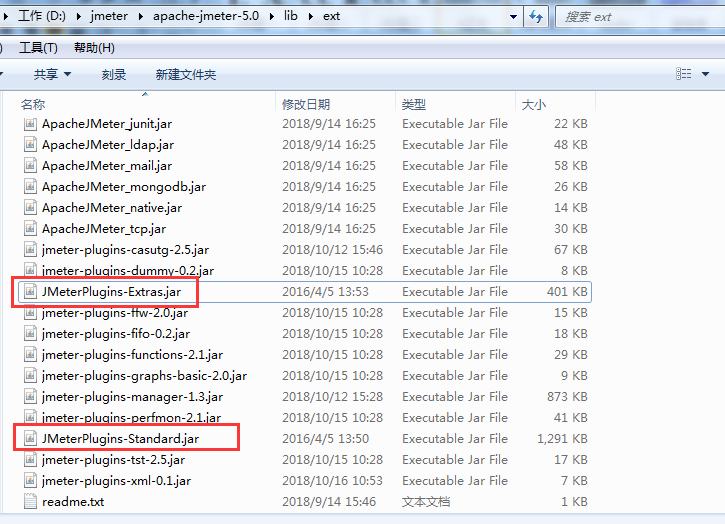
 如图上面两个是jmeter插件，可以再下面的链接中下载：

<https://jmeter-plugins.org/downloads/old>

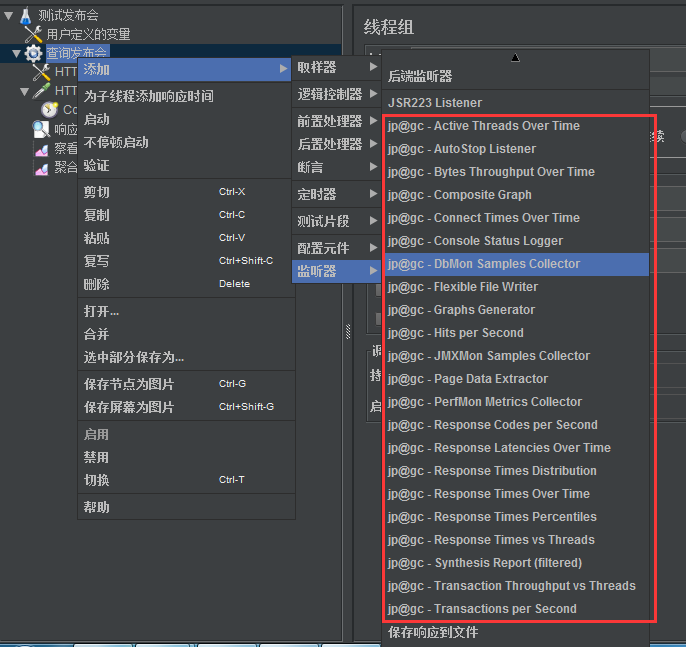
第三个需要通过其他途径下载

* + - 1. 解压压缩包

将JmeterPlugins-Extras.zip文件和JmeterPlugins-Standard.zip文件解压，将里边的.jar文件放在jmeter根目录的lib/ext目录下



如上图，把的两个jar包放到JMeter的 lib/ext目录下，重启jmeter，出现如下新增的组件，则说明启动成功



* + - 1. 常用组件简要介绍

**jp@gc - Bytes Throughput Over Time:**不同时间吞吐量展示（图表）   
聚合报告里，Throughput是按请求个数来展示的，比如说1.9/sec，就是每s发送1.9个请求；而这里的展示是按字节Bytes来展示的图表

**jp@gc - Composite Graph：**混合图表   
在它的Graphs里面可以设置多少个图表一起展示，它可以同时展示多个图表

**jp@gc - Hits per Second：**每秒点击量

**jp@gc - PerfMon Metrics Collector：**服务器性能监测控件，包括CPU，Memory，Network，I/O等等

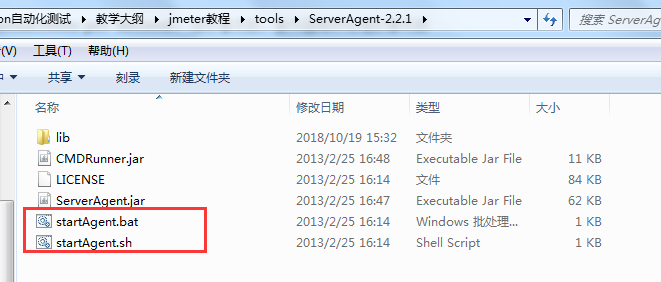
**jp@gc - Reponse Latencies Over Time：**记录客户端发送请求完成后，服务器端返回请求之前这段时间

**jp@gc - Reponse Times Distribution：** 显示测试的响应时间分布，X轴显示由时间间隔分组的响应时间，Y轴包含每个区间的样本数

**jp@gc - Transactions per Second：** 每秒事务数，服务器每秒处理的事务数

**jp@gc - Reponse Times vs Threads：展示事务响应时间与虚拟用户数之前的对应关系**

* + - 1. 将监控服务器的serverAgent拷贝到需监测的服务器

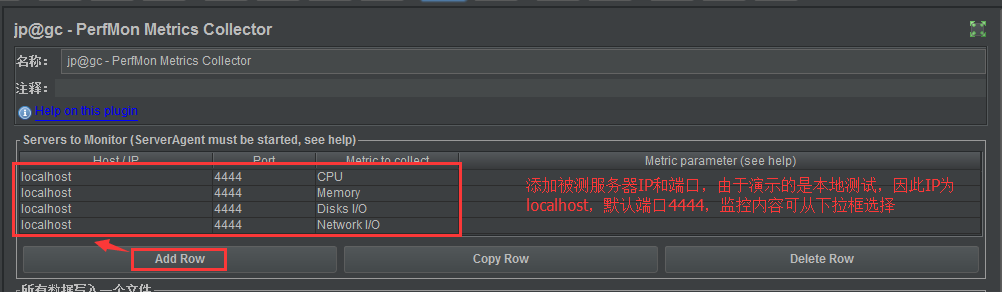


windows服务器中启动startAgent.bat，Linux服务器启动startAgent.sh即可

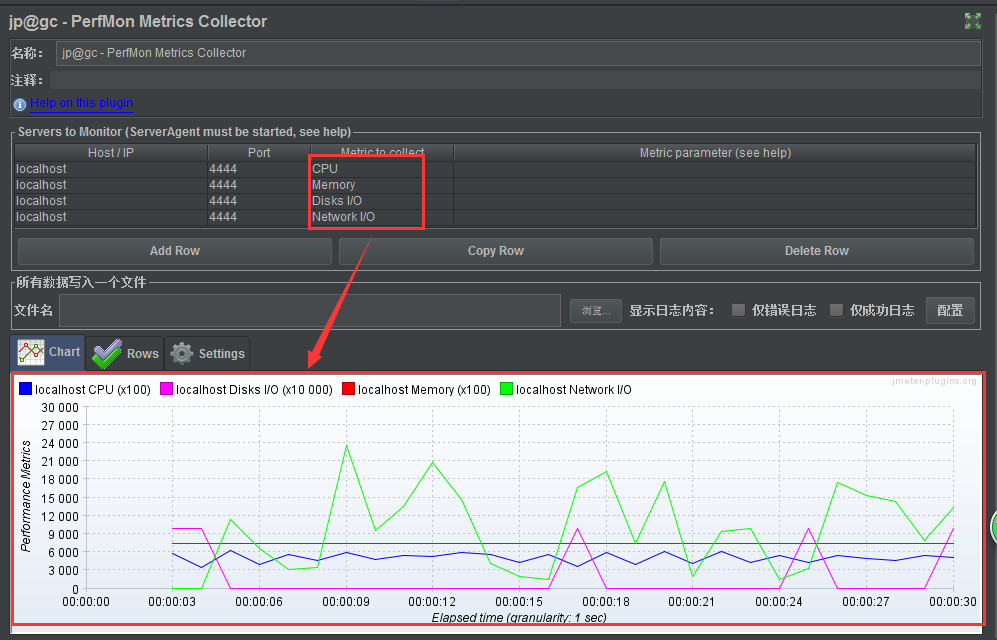
* + - 1. 准备测试脚本

以固定吞吐量请求为例

* + - 1. **配置监控服务器性能参数的组件**



* + - 1. 执行脚本查看结果



系统性能分析因素-CPU

CPU 在操作系统中是运行的根本,CPU的执行速度与性能好坏在很大程度上决定了系统整体的性能的快慢,最开始大部分任务是单线程调度,CPU在同一时间内只能运行一个线程,但随着多处理器技术的成熟,软件架构的深入和复杂度的提升,单处理系统组件在往多处理系统转变,这些都是利用处理器的多核心处理能力的特性来提升系统性能.在做系统性能分析前,首先我们要了解系统处理器的情况,如逻辑处理器,处理器型号,主频率,cache大小,是否支持超线程技术等信息,在知道这些的情况下,才能更好地进行我们系统的性能分析.

除此之外,CPU的使用率也是我们需要关注的很重要指标,当CPU处于满载状态时,很多时候我们要结合系统附带的一些系统监控分析工具,检查相关的系统日志,web服务器日志,DB日志等,结合辅助一些命令如top,free,uptime,sar等辅助分析为什么系统CPU会被完全占用,以及后续的解决优化方案.

系统性能分析因素—内存

在系统性能因素中,内存大小也是影响系统性能的一个非常核心的指标,当可用的内存太小,系统进程会被阻塞中,应用也将会变得非常缓慢,有时候会失去响应,严重的甚至可能会触发系统的OOM(内存溢出)从而引发应用程序被系统给杀死,更严重的情况可能会引起系统重启;当机器的内存太大的时候,有时候也是一种浪费,这时候我们可以考虑做一些缓存服务去提升系统性能.

虚拟内存也是在内存里面我们需要考虑的性能指标,在系统设计中,当系统的物理内存不够用的时候,就需要将物理内存中的一部分空间释放出来,以供当前运行的程序使用.那些被释放的空间可能来自一些长时间没什么操作的程序,这些被释放的空间被临时保存到虚拟内存空间中,等到那些程序要运行时,再从虚拟内存中恢复保存的数据到物理内存中.这样,系统总是在物理内存不够时,才进行内存之间的交换,有时可以越过系统性能瓶颈,节省系统升级费用.在做性能分析的时候,我们也要考虑系统有无设置虚拟内存,以及虚拟内存的使用情况.

内存分析

|  |  |
| --- | --- |
| **Mem:232600k,total** | **物理内存总量** |
| Mem:224688k,used | 使用的物理内存总量 |
| Mem:7912k,free | 空闲内存总量 |
| Mem:8508k,buffers | 用作内核缓存的内存量 |
| Mem:46264,cached | 缓冲的交换区总量 |
| **Swap:209714k,total** | **交换区总量** |
| Swap:62052k,used | 使用的交换区总量 |
| Swap:2035092k,free | 空闲的交换区总量 |

系统性能分析因素—网络

系统应用之间的交互,尤其是跨机器之间的,都是要基于网络的,因此网络宽带,响应时间,网络延时,阻塞等都是影响系统性能的因素.假如应用在不稳定和不安全的网络下会导致应用程序的超时,丢弃,阻塞,波动率大,这些在系统中都是不能接受的.我们需要一个可靠的,稳定的,能满足我们应用程序在机器之间畅通无阻地运行,这些需要测试工程师,网络管理员,系统管理员等一起合作把系统的网络完善.

在系统中,我们要考虑对应的网络是否可达,防火墙是否开启,端口访问,带宽是否有被限制,路由的寻址,网络的延时等问题.

系统性能分析因素—I/O

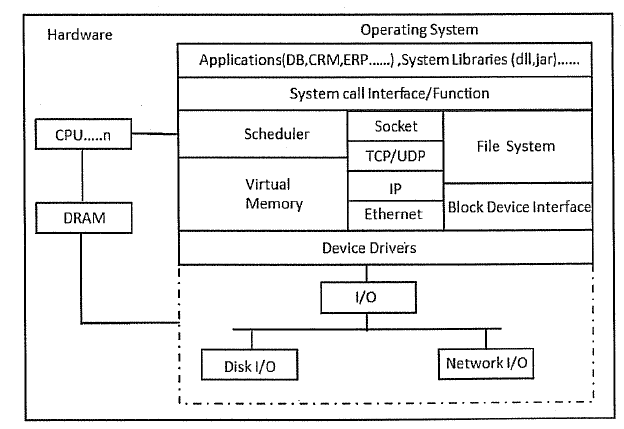
访问应用离不开系统的磁盘数据的读写,I/O读写的性能直接会影响系统程序的性能.读写的性能直接会影响系统程序的性能,磁盘I/O系统是系统中最慢的部分.I/O比较频繁的时候,如果I/O得不到满足会导致应用阻塞.针对I/O的场景模型,我们要考虑的有I/O的TPS,平均I/O数据,平均队列长度,平均服务时间,平均等待时间,IO利用率等指标.

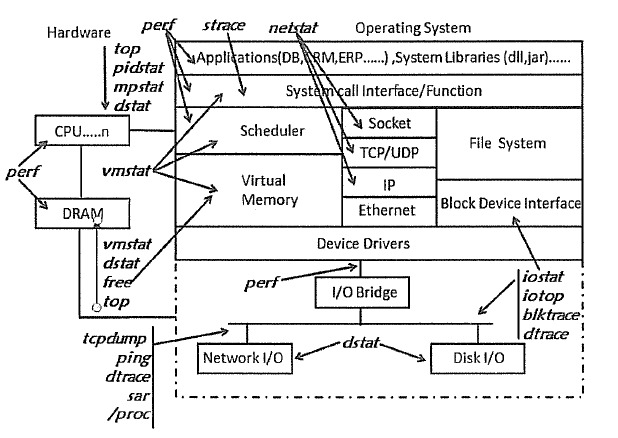
系统性能 分析因素—总结

上述所说的因素彼此之间是相互依赖的,任何一个因素处于高负载状态,都可能导致其他资源受到影响.比如;

1.大量的网络吞吐量会导致占用CPU资源增大,此时系统要分出部分资源去进行软件中断处理

2.大量的CPU开销会尝试更多的内存使用





CPU定位分析

|  |  |  |  |
| --- | --- | --- | --- |
| 模块 | 类型 | 度量方式 | 衡量标准 |
| CPU | 使用情况 | 1.vmstat 统计l-id 的计数  2.sar-u 统计l-%idle的计数  3.dstat命令 统计 l-idl的计数  4.mpstat-PALL 统计 l-%idle的计数  5.ps命令统计cpu的计数 | 注意≥50%  告警≥70%  严重≥90% |
| CPU | 满载 | 6.vmstat 的 r 计数 >CPU逻辑顺数  7.sar –q,”runq-sz”>CPU逻辑顺数  8.dstat –p,”run”>CPU逻辑顺数 | 运行的队列大于1时,证明已经有一定的负载了,不过这个计数也不绝对,需进一步分析其他的资源情况来断定CPU是否已满载负荷运作 |
| CPU | 错误 | 9.通过perf工具去捕获处理器的错误信息,需处理器支持 | 需处理支持 |
| 内存 | 使用情况 | 1.free命令查看使用情况  2.vmstat命令查看使用情况  3.sar-r命令查看使用情况  4.ps命令查看使用情况 | 注意≥50%  告警≥70%  严重≥90% |
| 内存 | 满载 | 5.vmstat的 si/so 比例辅助swapd 和free 利用  6.sar-W 查看每次缺页数  7.查看内核日志有无OOM 机制kill 进程  8.dmesg|grep killed | 1.so数值大,且swapd已经占比很高,内存肯定已经饱和  2.sar命令次缺页多意味已经在不停地和swap打交道,证明内存已经饱和  3.当内存不够用会触发内核的OOM机制 |
| 内存 | 错误 | 9.查看内有无physical failures  10.通过工具如valgrind 等进行检查 | 有计数 |
| 网络 | 使用情况 | 1.sar –n DEV 的收发计数大于网卡上限  2.ifconfig RX/TX 宽带超过网卡上限  3.cat/proc/net/dev/ 的速率超过上限  4.nicstat的util 基本满负荷 | 1.收发包的吞吐速率达到网卡上限  2.有延迟  3.有丢包  4.有阻塞 |
| 网络 | 满载 | 5.ifconfig dropped 有计数  6.nestat-s”segments retransimted”有计数  7.sar-n EDEV rxdrop txdrop 有计数 | 统计的丢包有计数证明已满载了 |
| IO | 使用情况 | 1.iostat –xz,”%util”  2.sar-d,”%util”  3.iotop的利用率很高  4.cat/proc/pid/sched|grep jowait | 注意≥50%  告警≥70%  严重≥90% |
| IO | 满载 | 5.iostat –xnzl,”avgqu-sz”>1  6.iostat await>70 | IO 已经有满载嫌疑 |
| IO | 错误 | 7.dmesg 查看io错误  8.smartctl/dev/sda | 有信息 |

Linux系统性能分析思路和实践

系统负载监控分析实践

uptime命令:主要用于获取主机运行时间和查询Linux系统负载等信息,uptime命令可以显示系统已经运行了多长时间,以及有多少用户登录,快速获取服务器的负荷情况,它信息显示依次为:现在时间,系统已经运行了多长时间,目前有多少登录用户,系统在过去的1分钟,5分钟,15分钟内的平均负载.

1.uptime的系统存活时间越长,意味着系统稳定,我们可以通过uptime查看系统这一段时间有无重启,这也是一种常见分析系统是否稳定的命令

2.通过uptime命令可以得知当前有多少登录用户,但相对来说w命令会更好地显示当前登录用户数的信息.

3.一般系统建议每个CPU内核的当前活动进程数量最好不要大于0.8,证明系统是空闲的,大于1且不小于3的时候,如果系统的其他资源很正常,那么系统的性能也可以接受的.但如果任务数大于5的话,那证明系统性能有问题了.以一个四核CPU的主机为例,当uptime的输出结果超过15,那就意味着当前系统负载非常严重,需要分析当前的进程调度策略,是否有阻塞等,估计此时可能打开运行脚本都会非常缓慢的.

4.系统负载的3个值表示的是系统过去的1分钟,5分钟,15分钟的一个平均值.通过这3个值的信息,我们 可以分析出系统负载的趋势:是否增加,稳固,降低等.

Linux系统性能分析思路和实践

负载uptime命令

CPU top命令

Windows 系统性能分析思路和实践

性能监视器综述:

perfmon 进入性能监视器

中间件Tomcat 监控 Probe

监控项

|  |  |  |
| --- | --- | --- |
| 类别 | 计数器 | 描述 |
| tomcat | JVM 内存 | 关注GC 回收频率,Full GC 次数越少越好 |
| 最大线程数 | 线程池连接数长期大80%以上,建议优化 |
| 数据库连接数 | 活动连接数长期大于80%以上,建议优化连接池 |
| 请求数  请求状态 | 线程数,线程状态,大量 Blocked 状态线程可以 Dump 线程栈信息进行分析 |

性能调优常用手段:

1.空间换时间,内存,缓存就是典型的空间换时间的例子.利用内存缓存从磁盘上取出的数据,CPU请求数据直接 从内存中获取,从而获取比从磁盘读取数据更高的效率

2.时间换空间:当空间成为瓶颈时,切分数据分批次处理,用更少的空间完成任务处理.上传大附件时经常用这种方式.

3.分而治之:把任务切分,分开执行,也方便并行执行来提高效率.

4.异步处理:业务链路上有任务时间消耗较长,可以拆分业务,减少阻塞影响.常见的异步处理机制有MQ(消息队列),目前在互联网应用中大量使用.

5.并行:用多个进程或者线程同时处理业务,缩短业务处理时间,比如我们在银行办理业务时,如果排队人数较多时,银行会加开柜台.

6.离用户更近一些:比如CDN技术,把用户的静态资源放在离用户更近的地方.

7.一切可扩展:业务模块化,服务化,良好的水平扩展能力.

分布式架构的运用给性能带来了革命性的提升,业务流程的调整也会显著提升系统性能,单系统的调优能够压榨出更高的处理能力.

系统的发展:

单体--🡪集群--🡪分布式--🡪分布式集群

单机性能分析与调优

客户端--🡪应用服务器--🡪数据库

我们的服务运行在中间件上,中间件与DB运行在操作系统上,操作系统来管理计算机硬件设备(CPU,内存,磁盘,网卡等设备).

单机性能分析流程:

Client:客户端

Load Machine:负载生成器—模拟用户负载

webserver:提供Web服务的服务器,即我们访问的web页面由此服务器提供服务;一般部署在Nginx,Apache等中间件上.

Middleware:中间件,比如Tomcat,Jboss,WebLogic等

OS:操作系统

System Resource:系统资源

APPserver:应用服务器,实现业务逻辑

DB:数据库服务器

配置优化:

JVM配置优化

连接池:

连接池配置参数

连接池配置多少连接合适

监控连接池

线程池:

缓存机制:

1. 课后使练习
   * + 1. 使用jmeter测试ECShop吞吐量
2. 面试题
   * + 1. 接口测试特点
       2. 性能测试参数
3. 每日一练
   * + 1. Python封装requests请求
       2. python封装获取接口返回值
4. 扩展知识或课外阅读推荐（可选）
   1. 扩展知识

(根据时间来选择是否讲解)

* 1. 课外阅读