# 新蜂商城性能测试方案

编写人：安杰

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档版本号 | 日期 | 作者 | 审核人 | 说明 |
| 1.0 | 2023-10-07 | 安杰 |  |  |
|  |  |  |  |  |

目录

[新蜂商城性能测试方案模板 1](#_Toc1537465184)

[1．背景 3](#_Toc1654106784)

[1.1 项目背景 3](#_Toc1396908273)

[1.2性能目标 3](#_Toc1546115307)

[2. 测试范围 3](#_Toc1007836049)

[2.1需要测试的特性 3](#_Toc1496951654)

[2.2不需要测试的特性 3](#_Toc1495524173)

[3. 准则 3](#_Toc1126171123)

[3.1启动准则 4](#_Toc1784683250)

[3.2结束准则 4](#_Toc1267285101)

[3.3 暂停/再启动准则 4](#_Toc517881561)

[4. 业务模型与性能指标 4](#_Toc284174436)

[4.1 业务模型/测试模型 4](#_Toc116114924)

[4.2 业务指标/性能指标 4](#_Toc1628376192)

[5. 系统系统架构图 4](#_Toc587061576)

[5.1 系统技术栈 4](#_Toc1204033514)

[5.2 系统逻辑架构图 4](#_Toc452864117)

[5.3 系统部署架构图 4](#_Toc605169451)

[6. 性能实施前提条件 4](#_Toc600410765)

[6.1 硬件环境 4](#_Toc78070102)

[6.2 工具准备 4](#_Toc11695997)

[6.2.1 测试工具 4](#_Toc1153609702)

[6.2.2 监控工具 4](#_Toc1235896398)

[6.3 数据准备 4](#_Toc1248927402)

[6.3.1 基础数据 4](#_Toc1217679636)

[7. 性能设计 5](#_Toc22486342)

[7.1 场景执行策略 5](#_Toc2118311769)

[7.1.1 场景递增策略 5](#_Toc1482001617)

[7.1.2 业务场景 5](#_Toc1485839013)

[7.2 监控设计 5](#_Toc1556444175)

[7.2.1 全局监控 5](#_Toc658945118)

[7.2.2 定向监控 5](#_Toc317430647)

[8. 项目组织架构 5](#_Toc707504981)

[9. 成果输出 5](#_Toc419262228)

[9.1 过程性输出 5](#_Toc646420189)

[9.2 结果输出 5](#_Toc264346350)

[9.2.1 性能性能测试报告 5](#_Toc1872922454)

[9.2.2 性能调优报告 5](#_Toc392386652)

[10. 项目风险分析 5](#_Toc2067663874)

[1.1    背景与 需求概述 6](#_Toc646354564)

[1.2    需求分析 6](#_Toc1308870622)

[1.2.1 测试对象分析 6](#_Toc1513547733)

[1.2.2 关键场景分析 6](#_Toc1252949816)

[2 环境概述 6](#_Toc102915030)

[2.1 线上 / 测试环境部署拓扑 6](#_Toc968573375)

[2.2 线上 / 测试环境硬件 8](#_Toc886669365)

[2.3 软件环境 8](#_Toc862991022)

[3 性能测试场景设计 8](#_Toc185554916)

[3.1 定时联盟数据同步 8](#_Toc475217768)

[4 性能测试计划 9](#_Toc493343583)

[5 风险评估 10](#_Toc191238414)

[5.1 技术风险 10](#_Toc1508488186)

[5.2 进度风险 10](#_Toc2116489267)

# 1．背景

## 1.1 项目背景

当面对日益增长的电商市场竞争和用户需求时，新蜂商城意识到了确保系统可靠性和提供优质用户体验的重要性。因此，针对新蜂商城进行性能测试，是为了评估其在不同负载下的性能表现以及系统的可扩展性。

## 1.2性能目标

1. 根据经典的电商下单流程，测试当前系统的单接口最大容量。
2. 根据业务比例设计容量场景，充分利用当前资源，找到当前系统的性能瓶颈，并优化，以达到系统的最佳运行状态。
3. 根据稳定性场景，判断当前系统可支持的系统最大累加容量。
4. 根据异常场景，判断当前系统中的异常对性能产生的影响。

# 测试范围

## 2.1需要测试的特性

电商主流程，如下所示：

|  |  |  |
| --- | --- | --- |
| 序号 | 操作 | 接口 |
| 1 | 打开首页 | /index |
| 2 | 用户注册 | /register |
| 3 | 用户登录 | /login |
| 4 | 设置默认地址 | personal/updateInfo |
| 5 | 浏览商品详情 | goods/detail/10907 |
| 6 | 添加购物车 | /shop-cart |
| 7 | 结账 |  |
| 8 | 支付 |  |
| 9 | 登出 |  |

## 2.2不需要测试的特性

批量业务

# 准则

## 3.1启动准则

1. 确定系统逻辑架构和部署架构和生产一致。
2. 确定基础数据和生产一致或按模型缩放。
3. 确定业务模型可以模拟生产真实业务。
4. 环境准备完毕，包括：
5. 功能验证通过。
6. 各组件基础参数梳理并配置正确。
7. 压力机到位，并部署完毕。
8. 网络配置正确，连接通畅，可以满足压力测试需求。
9. 测试计划、方案评审完毕。
10. 架构组、运维组、开发组、测试组及相关专家人员到位。

## 3.2结束准则

1. 达到项目要求的性能需求指标。
2. 关键性能瓶颈已解决。
3. 完成性能测试报告和性能调优报告

发送到发大发放

## 3.3 暂停/再启动准则

### 3.3.1暂停准则

1. 系统环境变化：举例：系统主机硬件损坏、网络传输时间超长、压力发生器出 现损坏、系统主机因别的原因需升级暂停等。
2. 测试环境受到干扰，比如服务器被临时征用，或服务器的其他使用会对测试结果造成干扰。
3. 需要调整测试环境资源，如操作系统、数据库参数等。
4. 该测试机型无法达到规划指标要求。
5. 出现测试风险中列出的问题。

### 3.3.1再启动准则

1. 测试中发现问题得以解决。
2. 测试环境恢复正常。
3. 测试风险中出现的问题已解决。
4. 环境调整完毕。

# 业务模型与性能指标

## 4.1业务模型/测试模型

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 操作 | 接口 | 业务比例 |
| 1 | 打开首页 | /index | 11% |
| 2 | 用户注册 | /register | 20% |

## 4.2 业务指标/性能指标

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 操作 | 接口 | 业务比例 | 目标tds | tps标准方差 | 响应时间 | 响应时间标准方差 |
| 1 | 打开首页 | /index | 11% | 100 | 5% | 200ms | 5% |
| 2 | 用户注册 | /register | 20% | 50 | 5% | 200ms | 5% |

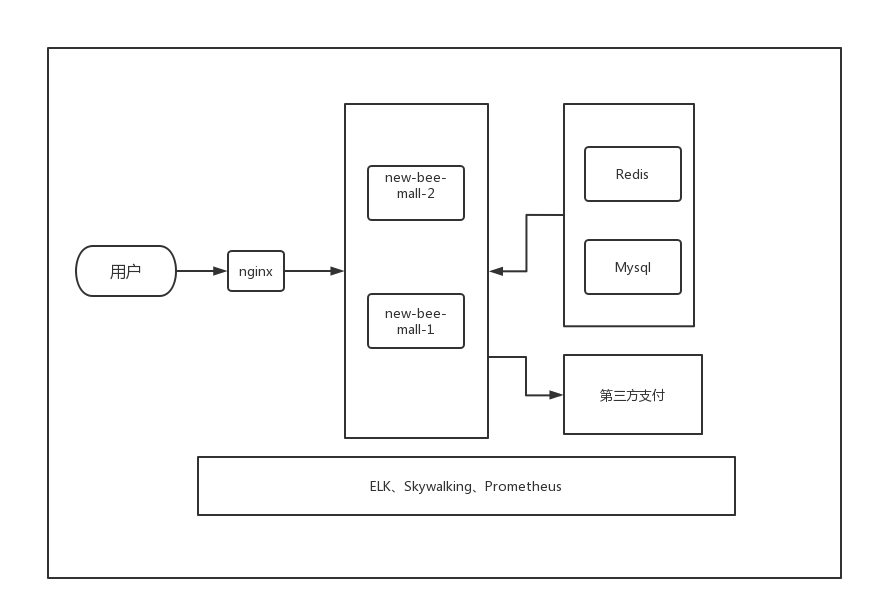
# 系统架构图

## 系统技术栈

|  |  |  |
| --- | --- | --- |
| 技术 | 说明 | 官网 |
| Jmeter | 压测引擎 | Jmeter.apache.org |
| Redis | 分布式缓存 | http://redis.io |
| ... |  |  |

备注：系统技术栈是让我们知道整个架构中用了哪些技术组件

## 系统逻辑架构图



备注：逻辑架构图是为了后续性能分析的时候，脑子里能有一个业务路径。我们在做性能分析时，要做响应时间的拆分，而只有了解了逻辑架构图才可以知道从哪里拆到哪里

## 系统部署架构图

备注：画部署架构图是为了让我们知道有多少节点、多少机器。在执行容量场景时，你的脑子里要有一个概念，就是这样的部署架构最大应该可以支持多少的容量上限。

# 6. 性能实施前提条件

## 6.1 硬件环境

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 主机 | 用途 | 配置 | 备注 |
| 1 | 物理服务器1 | kvm服务器 | 16c-32g-280g | [xn57014@shell.ceshiren.com ~]$ free -h  total used free shared buff/cache available  Mem: 3.7G 964M 127M 2.0G 2.6G 504M  Swap: 0B 0B 0B |
| 2 | 物理服务器2 | nfs服务器 | 4c-4g-300g |  |

备注：鉴于此，作为性能从业人员，我们必须要了解硬件配置和整体业务容量之间的关系

## 6.2 工具准备

### 6.2.1 测试工具

|  |  |  |
| --- | --- | --- |
| 序号 | 软件 | 版本 |
| 1 | jmeter | 5.3 |
| 2 | influxDB | 1.8 |
| 3 | grafana | 7.1.0 |

在测试过程中，我们将使用 JMeter 的 backend Listener 把数据直接发到 InfluxDB 中，然后再由 Grafana 来展现。我们不使用 JMeter 的分布式执行功能或本地收集数据的功能，因为这样会消耗本地的 IO。

### 6.2.2 监控工具

|  |  |  |
| --- | --- | --- |
| 序号 | 软件 | 功能 |
| 1 | Promethues | 通过Promethues提供的Exporter功能采集数据。包括k8s的各个组件、centos、mq、redis、myslq、redis、nfs等组件 |
| 2 | Grafana | 展示Promethues采集的数据 |
| 3 | Skywalking | java应用apm链路监控工具 |
| 4 | Elasticsearch | 存储各类日志 |
| 5 | Kibana | 展示各类日志 |
| 6 | Logstash | 采集各类日志 |

## 6.3 数据准备

### 6.3.1 基础数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 数据类型 | 表 | 准备前数据 | 准备后数据 |
| 1 | 用户信息 | Ums\_member | 10 | 2345655 |
| 2 | 地址信息 |  | 10 | 45655 |
| 3 | 商品数据 |  | 10 | 5533 |
| 4 | 订单表 |  | 10 | 234655 |

基础数据要满足两个特性：

1. 满足生产环境的真实数据分布
2. 参数化数据一定要使用基础数据来覆盖真实用户

# 7. 性能设计

## 7.1 场景执行策略

### 7.1.1 场景递增策略

对于性能场景，我一直在强调一个观点，那就是性能的场景必须满足两个条件：

1. 连续
2. 递增

### 7.1.2 业务场景

#### 基准场景

基准场景必须是容量场景的前奏。具体怎么做呢？那就是在基准场景中，我们也要通过递增连续的场景做到最大 TPS。也就是说在基准场景中，我们要把单接口或单业务压到最大 TPS，然后来分析单接口或单业务的瓶颈点在哪里。

可能你会问，在基准场景中有没有必要做调优的动作呢？

根据我的经验，应该先判断当前单接口或单业务的最大 TPS，有没有超过目标 TPS。如果超过，并且响应时间也在业务可接受的范围之内，那就不用调优。如果没有超过，那必须要做调优。

#### 容量场景

有了基准场景的结果之后，我们就进入了容量场景的阶段。在容量场景中，我们还是要继续秉承“连续、递增”的执行思路，最重要的是，要实现我们前面提到的业务模型，来真实模拟线上的业务场景。

#### 稳定性场景

在稳定性场景中，我们只有两个关键点：

1. 第一个关键点：稳定性场景的时长。
2. 第二个关键点：用多大的 TPS 来执行。

#### 异常场景

对于常规的异常场景，我们经常做的就是：宕主机；宕网卡；宕应用。

## 7.2 监控设计

### 7.2.1 全局监控

其实，有了前面的监控工具部分，监控设计就已经出现在写方案之人的脑子里了。对于我们这个课程所用的系统，全局监控如下所示：

从上图来看，我们使用 Prometheus/Grafana/Spring Boot Admin/SkyWalking/Weave Scope/ELK/EFK 就可以实现具有全局视角的第一层监控。对工具中没有覆盖的第一层计数器，我们只能在执行场景时再执行命令来补充了。

### 7.2.2 定向监控

那后面的定向监控怎么办呢？在这里我也大体列一下常用的工具。不过，请你注意，这些工具是在有问题的时候才会去使用。

|  |  |  |
| --- | --- | --- |
| 序号 | 工具 | 作用 |
| 1 | Jstack  Jmap  Jvisualvm  Arthas | 监控Java应用。这里的工具各有各的使用场景 |
| 2 | Mysqlreport  Pt-digest-query | 分析mysql |
| 3 | Perf  Strace | 跟踪操作系统级别的调用 |

# 项目组织架构

# 9. 成果输出

## 9.1 过程性输出

1. 脚本
2. 场景执行结果
3. 监控结果
4. 问题记录

建议：在性能项目中，尽量多做一些归档整理的工作，以备在后面的项目中查阅，并实现自己的技术积累

## 9.2 结果输出

### 9.2.1 性能测试报告

1. 性能结果报告一定要有结论，而不是给出一堆“资源使用率是多少”、“TPS 是多少”、“响应时间是多少”这种描述类的总结语。你想想，性能结果都在这个报告中了，谁还看不见怎么滴？还要你复述一遍吗？我们要给出“当前系统可支持 XXX 并发用户数，XXX 在线用户数”这样的结论
2. 一定不要用“可能”、“或许”、“理应”这种模棱两可的词，否则就是在赤裸裸地耍流氓
3. 性能结果报告中要有对运维工作的建议，也就是要给出关键性能参数的配置建议，比如线程池、队列、超时等。
4. 性能结果报告中要有对后续性能工作的建议

### 9.2.2 性能调优报告

为什么我要强调单独写调优报告呢？因为调优报告才是整个性能项目的精华，调优报告中一定要记录下每一个性能问题的问题现象、分析过程、解决方案和解决效果。可以说，调优报告完全是一个团队技术能力的体现。

# 10. 项目风险分析

比较常见的风险：

1. 业务层的性能需求不明确
2. 环境问题
3. 数据问题
4. 业务模型不准确
5. 团队间协调沟通困难
6. 瓶颈分析不到位，影响进度
7. ......