ISTQB®软件测试人员认证

基础级大纲
性能测试

2018 版

国际软件测试认证委员会

本大纲由 **ASTQB** 和 **GTB** 提供

中文版的翻译编辑和出版统一由 **ISTQB®**授权的 **CSTQB®**负责

英文版权声明

中文版权声明

性能测试工作组：

Graham Bath

Rex Black

Alexander Podelko

Andrew Pollner

Randy Rice

版本历史

版本	日期	说明
Alpha V04	2016 年 12 月 13 日	纽约会议前版本
Alpha V05	2016 年 12 月 18 日	纽约会议后版本
Alpha V06	2016 年 12 月 23 日	重构第 4 章,按照纽约会议商定重新编号及调整
Alpha V07	2016 年 12 月 31 日	增加作者备注
Alpha V08	2017 年 2 月 12 日	Alpha 预版
Alpha V09	2017 年 4 月 16 日	Alpha 预版
Alpha Review V10	2017 年 6 月 28 日	准备 Alpha 版本评审
V2017v1	2017 年 11 月 27 日	Alpha 版本评审
V2017v2	2017 年 12 月 15 日	Alpha 版本更新
V2017v3	2018 年 1 月 15 日	技术编辑
V2017v4	2018 年 1 月 23 日	术语校对
V2018 b1	2018 年 3 月 1 日	候选 Beta 版
V2018 b2	2018 年 3 月 17 日	Beta 版本发布
V2018 b3	2018 年 8 月 25 日	合并 Beta 版本备注
Version 2018	2018 年 12 月 9 日	ISTQB®成员大会发布版本

中文版本	日期	说明
0.2	08.06.2019	CSTQB®性能测试工作组翻译
0.4	12.12.2019	CSTQB®性能测试工作组评审修订
1.0	29.01.2020	CSTQB®正式发布

目录

版本历史 3
目录 4
致谢 6
0. 课程大纲引言 7
0.1 目的 7
0.2 基础级性能测试认证 7
0.3 收益 7
0.4 考试学习目标 8
0.5 培训时间建议 8
0.6 报考要求 8
0.7 信息资料来源 8
1. 基本概念 – 60 分钟 9
1.1 性能测试原则 9
1.2 性能测试类型 10
1.3 性能测试的测试类型 11
1.3.1 静态测试 11
1.3.2 动态测试 11
1.4 负载生成的概念 11
1.5 常见性能效率失效模式及原因 12
2. 性能测量基础 – 55 分钟 14
2.1 性能测试中收集的典型度量 14
2.1.1 为何需要性能度量 14
2.1.2 收集性能测量和度量 14
2.1.3 选择性能度量 16
2.2 汇总性能测试的结果 16
2.3 性能度量的关键来源 16
2.4 性能测试的典型结果 17
3. 软件生命周期中的性能测试 – 195 分钟 18
3.1 性能测试的主要活动 18
3.2 不同架构的性能风险类型 19
3.3 软件开发生命周期的性能风险 20
3.4 性能测试活动 21
4. 性能测试任务 – 475 分钟 23
4.1 策划 23
4.1.1 推导性能测试目标 23
4.1.2 性能测试计划 24
4.1.3 关于性能测试的沟通 27
4.2 分析、设计与实施 28
4.2.1 典型通信协议 28
4.2.2 事务处理 28
4.2.3 识别操作配置文件 29
4.2.4 创建负载配置文件 30
4.2.5 分析吞吐量和并发性 32

4.2.6 性能测试脚本的基本结构 32
4.2.7 实现性能测试脚本 33
4.2.8 准备执行性能测试 34
4.3 执行 36
4.4 分析结果及报告 37
5. 工具 – 90 分钟 40
5.1 工具支持 40
5.2 工具适用性 41
6. 参考资料 42
6.1 标准 42
6.2 ISTQB® 文档 42
6.3 参考书 42
7. 索引 43

致谢

本文件由美国软件测试委员会（ASTQB）和德国测试委员会（GTB）联合编制：

Graham Bath (GTB 工作组组长)
Rex Black
Alexander Podelko (CMG，计算机测试小组)
Andrew Pollner (ASTQB 工作组组长)
Randy Rice

核心团队成员向评审团队付出的努力和提出的建议表示感谢。ASTQB 向计算机测试小组（CMG）为制定基础大纲所做的贡献表示感谢。

以下成员参与了评审、评论和大纲表决工作：

Dani Almog	Marek Majernik	PéterSótér
Sebastian Chece	Stefan Massonet	Michael Stahl
Todd DeCapua	Judy McKay	Jan Stiller
WimDecoutere	Gary Mogyorodi	Federico Toledo
FransDijkman	JoernMuenzel	Andrea Szabó
Jiangru Fu	Petr Neugebauer	YaronTsubery
Matthias Hamburg	Ingvar Nordström	Stephanie Ulrich
ÁgotaHorváth	MeilePosthuma	MohitVerma
MiekeJungebload	MichaëlPilaeten	Armin Wachter
Beata Karpinska	Filip Rechtoris	Huaiwen Yang
GáborLadányi	Adam Roman	Ting Yang
Kai Lepler	Dirk Schweier	
IneLutterman	Marcus Seyfert	

英语版大纲由 ISTQB®大会于 2018 年 12 月 9 日正式发布。

本课程大纲中文版翻译参与者（按姓氏拼音排序）

范兴元、黄金虎、黄钦、金音前、李春慧、李欣、商超博、陶显锋、熊晓虹（组长）、杨洪梅、张大健。

本课程大纲中文版 QA 评审参与者：任亮

致谢企业：广州永融科技股份有限公司

0. 课程大纲引言

0.1 目的

本大纲根据国际软件测试认证委员会对基础级性能测试的要求进行编写。美国软件测试认证委员会（ASTQB ）和德国测试委员会（GTB）提供此大纲的主要目的：

1. 各个成员国认证委员会需将大纲翻译成当地语言并分发给培训机构，并可以根据特定语言对大纲进行适度润色、修改以保证语句通顺可读。
2. 考试委员会可根据特定语言，按照大纲的学习目标设计考题。
3. 培训机构需根据大纲准备课程并选择最适宜的教学方法。
4. 需要认证的考生，根据大纲准备考试（作为培训课程的一部分或独立使用）。
5. 国际软件和系统工程领域，应以此大纲为基础，推进软件和系统测试的蓬勃发展，并以此为基础著书和发表文章。

ASTQB 和 GTB 允许其他组织、机构在获得书面授权后使用此大纲内容。

0.2 基础级性能测试认证

基础级资格认证的目标对象是软件测试工作相关的、希望拓宽性能测试知识的人员，或希望开始专职从事性能测试的人员。资格认证还针对任何参与性能工程，希望更好地了解性能测试的人员。

基础级性能测试大纲主要考虑以下几个方面：

- 技术方面
- 方法方面
- 组织方面

ISTQB®高级测试技术分析师大纲[ISTQB®_ALTTA_SYL]中关于性能测试的内容与本大纲一致并由此延伸。

0.3 收益

本部分列举了获得基础级性能测试认证者可预期的收益。

PTFL-1 理解性能效率和性能测试的基本概念
PTFL-2 定义性能风险、目标和需求，以满足利益干系人的要求和预期
PTFL-3 理解性能度量指标及收集办法
PTFL-4 为实现既定目标和需求制定性能测试计划
PTFL-5 设计、实施和执行基本性能测试
PTFL-6 分析性能测试结果并向不同利益干系人陈述影响
PTFL-7 向不同利益干系人解释性能测试过程、原理、结果及影响
PTFL-8 理解性能测试工具的种类、用途及选择准则
PTFL-9 确定性能测试活动如何与软件生命周期保持一致

0.4 考试学习目标

学习目标是收益的前提，同时也被作为基础级性能测试认证考试题目编写的基础。学习目标具有不同的认知水平要求（K-等级）。

K-等级，或者说认知水平等级，是根据布鲁姆 [安德森 2001]修订的分类方法来对学习目标进行分类。ISTQB® 用这种分类方法设计考试大纲。

本大纲包括如下 4 个 K-等级（K1 到 K4）：

K-等级	关键字	描述
1	牢记	考生需要牢记和回忆相关术语或概念
2	理解	考生应能够对大纲主题相关的内容进行解释和分析
3	应用	考生应可以选择正确的测试概念或者技术，并应用到给定的场景中
4	分析	考生应可以将与流程或技术有关的信息分成不同组成部分，以便更好的理解，可以对现象和结论进行划分

通常来讲，本大纲所有章节都是 K1 内容，考生需要识别、牢记、回想起相关术语或概念。K2，K3 和 K4 级的学习目标会在后面的章节之初出现。

0.5 培训时间建议

本大纲定义了每个学习目标的最小培训时长，每章建议的总授课时间标记在章节标题后。

培训机构需注意，其他 ISTQB® 大纲根据不同的 K 等级分配了固定的标准培训时长。本性能测试大纲无需严格采用这种形式。培训机构可根据每一学习目标的最小培训时长选择更灵活更实际的培训时长。

0.6 报考要求

获得 ISTQB® 基础级核心模块认证的考生方可报名参加基础级性能测试认证。

0.7 信息资料来源

本课程大纲中使用的术语，在 ISTQB® 软件测试术语表[ISTQB®_GLOSSARY]文档中做了详细定义。

第六章节列出了一些关于性能测试的推荐书籍和文章。

1. 基本概念– 60 分钟

关键词

容量测试，并发测试效率，耐力测试，负载生成，负载测试，性能测试，可扩展性测试，峰值测试，压力测试

基本概念的学习目标

1.1 原则和概念

PTFL-1.1.1 (K2) 理解性能测试的原则

1.2 性能测试类型

PTFL-1.2.1 (K2)理解不同类型的性能测试

1.3 性能测试中的测试类型

PTFL-1.3.1 (K1) 牢记性能测试中的测试类型

1.4 负载生成的概念

PTFL-1.4.1 (K2) 理解负载生成的概念

1.5 性能测试中常见的失效及其原因

PTFL-1.5.1 (K2) 举例说明性能测试中常见的失效模式及其原因

1.1 性能测试原则

性能效率（或简称“性能”），是为在各种固定及移动平台上使用应用程序的用户提供“良好体验”的一个关键部分。在为最终用户建立可以接受的质量水平时，性能测试具有非常关键的作用，通常与其他方法紧密联合，比如易用性工程和性能工程。

并且，在加入负载的条件下（例如在性能测试期间）去评估功能的适合性、易用性和其他质量特性，可能会发现特定负载下的问题，影响到这些质量特性。

性能测试并不局限在关注终端用户的 Web 领域。它也和多种系统架构下的不同应用领域相关，比如传统的客户端-服务器架构，分布式应用和嵌入式应用。从技术上讲，性能效率在标准 ISO 25010[ISO 25000]产品质量模型中被归类为非功能性质量特征，包括下面的三个子特征。选择适当的重点和优先级取决于风险评估，以及不同利益干系人的需要。测试结果分析还可能发现需要解决的其他风险区域。

时间特性：一般来说，评估时间特性是性能测试最常见的目的。这项测试检查一个组件或系统在特定时间范围和条件下响应用户或系统输入的能力。时间特性的度量有多种方式，如系统响应用户输入的“端到端”时间，或是一个软件组件执行某项任务使用的 CPU 周期数量。

资源利用：如果系统资源的可用性被认为是一项风险，对这些资源的利用（例如对有限内存的分配）需要通过执行特定的性能测试来调查。

容量：如在所需系统容量极限（如用户数量数据量）内的系统行为问题被认为是一项风险，则需要执行性能测试来评估系统架构的适用性。

性能测试通常采用实验的形式，这样可以对特定的系统参数进行测量和分析。测试可以迭代地进行，以支持系统分析、设计和实施，从而能够做出架构决策，并帮助确定利益干系人的期望。

下面的性能测试原则特别需要注意：

- 性能测试必须与不同利益干系人的期望保持一致，特别是用户、系统设计者和操作人员。
- 测试必须是可重复的。在一个不变的系统上重复测试，必须得到统计上一致的结果（在允许误差范围内）。
- 测试必须产生可以理解的结果，以便于与利益干系人的期望比较。
- 在资源运行的情况下，测试可以在代表生产系统的完整或部分的系统或测试环境上执行。
- 测试必须在项目规定的时间范围内是切实可行的。

参考书籍[Molyneaux09] and [Microsoft07]　对性能测试的这些准则和实践提供了详细的背景介绍。

上述 3 个质量子特征都会影响被测系统的规模扩展性。

1.2 性能测试类型

性能测试有不同的类型，可以根据测试目标应用到特定的项目。

性能测试
性能测试是一个包括多种类型测试的概括性术语，关注点是系统或组件在不同负载下的性能（响应性）。

负载测试
负载测试关注的是系统处理不断增加的真实负载的能力。这些负载通过可控数量的并发用户或者进程来产生。

压力测试
压力测试关注的是系统或组件处理超出预期或特定值的峰值负载的能力。也可以用于评估系统在资源缺少时的处理能力，比如可用的计算能力、带宽和内存。

可扩展性测试
可扩展性测试关注的是系统满足未来效率需求的能力，这可能超出了现在的需求。测试的目标是在不违反当前的性能需求也不产生失效的前提下，确定系统的增长能力（例如服务更多用户，或存储更多数据）。一旦确定了可扩展性的极限，就可以在生产中设置和监控阈值，在可能出现问题时发出警告。另外也可以用适当调整生产环境的硬件数量。

峰值测试
峰值测试关注的是系统对突然爆发的峰值负载做出正确反应并随后恢复到稳定状态的能力。

耐力测试
耐力测试关注的是系统在特定运行环境下一段时间内的稳定性。这种测试验证的是资源不足问题（例如内存泄漏、数据库连接，线程池）等会导致系统在某个拐点性能下降或者产生失效。

并发测试
并发测试关注的是特定操作同时发生的影响（例如大量用户同时登陆）。并发问题很难找到和重现，特别是在很难测试或无法控制的环境中，比如生产环境。

容量测试
容量测试检验的是系统在满足规定的性能目标前提下，可以支持多少用户或者事务数量。这些目标也可以通过事务产生的数据量来确定。

1.3 性能测试的测试类型

性能测试中用到的主要测试类型包括静态测试和动态测试。

1.3.1 静态测试

相比功能适用性测试，性能测试中静态测试活动往往更加重要。因为很多严重的性能缺陷是在系统架构和系统设计阶段引入的。这些缺陷可能由于设计者和架构师的误解或者缺乏(相关)知识而产生，也可能因为需求没有充分捕捉到响应时间、吞吐率、资源利用目标、预期负载和用途、限制条件等等要素。

性能测试的静态测试活动可以包括：

- 关注性能及性能风险的需求评审
- 数据库架构，实体关系图，元数据，存储过程，查询的评审
- 系统和网络架构评审
- 系统关键部位的代码评审（如复杂算法）

1.3.2 动态测试

当系统已经构建起来，动态性能测试应该尽早开始。动态性能测试的时机包括：

- 在单元测试期间，包括使用信息分析来确定潜在瓶颈，使用动态分析来评估资源利用。
- 在组件集成测试期间，贯穿（跨组件）关键用例和工作流，尤其是在集成不同用例功能时，或者与工作流的主干结构集成时。
- 在系统测试期间，在不同负载条件下检查总体端到端表现。
- 在系统集成测试期间，特别是对关键系统间接口的数据流和工作流。这个阶段“用户”是另一个系统或机器（例如传感器或其他系统输入）并不少见。
- 在验收测试阶段，建立用户、客户、操作员对系统性能的信心，以及在真实条件下对系统进行调优（但这时通常不是为了发现系统中的性能缺陷）。

在更高的测试级别（如系统测试和系统集成测试）中，使用真实的环境、数据和负载对于准确的结果至关重要（见第 4 章）。在敏捷和其他迭代增量生命周期中，团队应该将静态和动态性能测试纳入早期迭代，而不是等待最终迭代来解决性能风险。

如果定制硬件或新硬件是系统的一部分，则可以使用模拟器执行早期的动态性能测试。但是，最好尽快在实际硬件上开始测试，因为模拟器通常不能充分捕获资源约束和与性能相关的行为。

1.4 负载生成的概念

为了进行第 1.2 节中描述的各种类型的性能测试，必须对代表性系统负载进行建模、生成并提交给被测系统。负载可与功能测试用例中使用的数据输入类似，但在以下主要方面有所不同：

- 性能测试负载必须表示多个用户输入，而不仅仅是一个
- 性能测试负载可能需要专用的硬件和工具来生成
- 性能测试负载的生成取决于被测系统中不存在任何可能影响测试执行的功能缺陷。

在进行性能测试时，有效且可靠地生成指定负载是一个关键的成功因素。负载生成有不同的方法。

使用用户界面生成负载
如果只需要代表一小部分用户，并且可以使用所需数量的软件客户端来执行所需的输入，这可能是一种适当的方法。这种方法也可以与功能测试执行工具结合使用，但是随着要模拟的用户数量的增加，可能会很快变得不实用。用户界面（UI）的稳定性也是一个关键的依赖关系。频繁的更改会影响性能测试的可重复性，并且可能对维护成本产生显著影响。通过 UI 进行测试可能是端到端测试最具代表性的方法。

使用众测生成负载
这种方法依赖于大量测试人员，他们将代表真正的用户。在众测中，测试人员被组织起来，这样就可以生成所需的负载。这可能适合测试从世界各地都可以访问的应用（例如基于 Web），并且可能涉及用户从各种不同的设备类型和配置生成负载。虽然这种方法可以引入大量的用户，但生成的负载将不会像其他选项那样具有可重复性和精确性，而且组织起来更复杂。

使用 API 生成负载
这种方法类似于使用 UI 进行数据输入，它使用应用程序的 API 而非 UI 来模拟用户与被测系统的交互。因此，该方法对用户界面中的更改（例如延迟）不太敏感，并且事务处理的方式可以和直接通过 UI 输入的方式相同。可以创建专用脚本重复调用特定的 API，与使用 UI 输入相比，可以模拟更多的用户。

使用捕获的通信协议生成负载
这种方法包括在通信协议级别捕获用户与被测系统的交互，然后重播这些脚本，以可重复和可靠的方式模拟潜在的大量用户。第 4.2.6 和 4.2.7 节描述了这种基于工具的方法。

1.5 常见性能效率失效模式及原因

当然，动态测试过程中可以发现许多不同的性能故障模式，以下是一些常见故障（包括系统崩溃）的示例以及典型原因：

在所有负载水平下响应缓慢
在某些情况下，无论负载如何，响应都是不可接受的。这可能是由底层性能问题引起的，包括但不限于糟糕的数据库设计或实施、网络延迟和其他后台负载。这些问题可以在功能性和可用性测试中发现，而不仅仅是性能测试，因此测试分析师应密切关注并报告它们。

中高负载下反应缓慢
在某些情况下，响应会随着中到重度负载而降低，即使这些负载完全在正常、预期和允许的范围内。这是不可接受的。潜在缺陷包括一个或多个资源的饱和及后台负载变化。

随着时间的推移，响应降低
在某些情况下，随着时间的推移，响应会逐渐的或严重的下降。根本原因包括内存泄漏、磁盘碎片、随时间增加的网络负载、文件存储库的增长以及意外的数据库增长。
.
高负载或超高负载下出错处理不充分或粗暴
在某些情况下，响应时间是可以接受的，但出错处理在高负载和超出极限负载水平下会降低。潜在缺陷包括资源池不足、队列和堆栈太小以及超时设置太快。

上述常见故障类型的具体示例包括：

- 提供公司服务信息的基于 Web 的应用程序在七秒钟内未响应用户请求（一般行业经验法则）。在特定的负载条件下无法达到系统的性能效率。
- 在突然出现大量用户请求（例如，重大体育赛事的门票销售）时系统崩溃或无法响应用户输入。处理该用户数量的系统容量不足。
- 当用户提交对大量数据的请求（例如，在网站上发布一份大型而重要的报告以供下载）时，系统响应会显著降低。处理产生大量数据的系统容量不足。

- 在需要在线处理之前，无法完成批处理。批处理的执行不能在允许时间范围内完成。
- 当并行进程对动态内存产生巨大需求又无法及时释放时，实时系统会耗尽内存。内存大小不够，或者内存请求的优先级处理不当。
- 向实时系统组件 B 提供输入的实时系统组件 A 无法按要求的速率计算更新。整个系统无法及时响应，可能会失败。必须评估和修改组件 A 中的代码模块（“性能分析”），以确保能够达到要求的更新率。

2. 性能测量基础 – 55 分钟

关键词
测量、度量

性能测量基础的学习目标

2.1 性能测试中收集的典型度量
PTFL-2.1.1 (K2) 理解性能测试中收集的典型度量

2.2 汇总性能测试的结果
PTFL-2.2.1 (K2) 解释汇总性能测试结果的原因

2.3 性能度量的关键来源
PTFL-2.3.1 (K2) 理解性能度量的关键来源

2.4 性能测试的典型结果
PTFL-2.4.1 (K1) 回顾性能测试的典型结果

2.1 性能测试中收集的典型度量

2.1.1 为何需要性能度量

准确的测量和从这些测量中得出的度量指标对于定义性能测试的目标和评估性能测试的结果至关重要。如果没有预先了解需要哪些测量和度量指标，就不应进行性能测试。若忽略此建议，则会产生以下项目风险：

- 性能水平是否能达到满足运行目标的可接受水平不确定
- 性能需求并未用可测量的术语定义
- 可能无法识别趋势，预言性能水平低下
- 无法将现有的性能测试结果与性能测量基准线比对，评估性能是可接受和/或不可接受
- 根据一个或多个人的主观意见来评估性能测试结果
- 无法理解性能测试工具所提供的结果

2.1.2 收集性能测量和度量

与任何形式的测量一样，可以用精确的方式获得并表达度量。因此，本节中描述的任何度量和测量可以并应该定义为有特定的上下文意义。这个问题关乎执行初始的测试和了解哪些度量需要进一步细化，哪些需要添加。

例如，响应时间的度量可能包含在任何一组性能指标中。但是，出于有意义且可行性方面的考虑，需进一步定义时刻、并发用户数、正在处理的数据量等。

在一个具体性能测试中，收集的度量可能基于以下方面而有所不同：

- 业务环境（业务过程、客户和用户行为、利益干系人期望）

- 运行环境（技术以及技术的使用方式）
- 测试目标

例如，一个国际电子商务网站性能测试选择的度量与一个用于控制医疗设备功能性的嵌入式系统的性能测试所选择的度量必然不同。

一个用于对性能的测量和度量进行分类的常用方法是：考虑对性能进行评估的技术环境、业务环境或运行环境。

下述的测量和度量种类是性能测试中获取的常见种类。

技术环境

性能度量依据技术环境的类型不同而有所不同，如下表所示：

- 基于 Web
- 移动端
- 物联网
- 桌面客户端设备
- 服务器端处理
- 大型机
- 数据库
- 网络
- 环境中所运行软件的固有特性（如，嵌入式）

包括下列度量指标：

- 响应时间（如每个事务的响应时间、每个并发用户的响应时间、页面加载时间）
- 资源利用情况（如 CPU、内存、网络带宽、网络延迟、可用磁盘空间、I/O 速率、空闲和繁忙线程）
- 关键事务吞吐率（即在一个特定时间周期内，可以处理的事务数量）
- 批处理时间（如等待时间、吞吐量时间、数据库响应时间、任务完成时间）
- 影响性能的错误数量
- 任务完成时间（如创建、读取、升级以及删除数据）
- 共享资源的后台加载（特别是在虚拟化环境中）
- 软件度量（如代码复杂度）

业务环境

从业务或功能视角，可以包括下列性能度量：

- 业务处理效率（如一个完整业务过程的执行速度，包括正常、备用、以及异常的用例流程）
- 数据、交易、以及其他工作执行单元的吞吐量（如每小时订单处理量、每分钟的数据行增加量）
- 服务水平协议（SLA）的符合或违反率（如单位时间 SLA 的违反量）
- 使用的范围（如在指定时间，执行任务的全球或本国用户百分比）
- 使用的并发情况（如并发执行一个任务的用户数）
- 使用时段（如在峰值负载期间处理的订单数）

运行环境

性能测试的运行方面侧重于那些通常被认为本质上非面向用户的任务。包括以下这些内容：

- 运行过程（如环境启动、备份、关机、恢复所需时间）
- 系统恢复（如从一个备份中恢复数据所需的时间）
- 警报和警告（如系统发出一个警报和警告所需时间）

2.1.3 选择性能度量

应当注意的是，收集过度的度量并不一定是件好事。选定的每个度量都需要一种一致的收集和报告方法。定义一组支持性能测试目标的、可获得的度量集是非常重要的。

例如，目标-问题-度量（GQM）方法是将度量与性能目标保持一致的有效方法。思路是首先确定目标，然后提出问题以确定目标何时实现。度量与每个问题相关联，以确保问题的答案是可测量的。（参见专家级大纲的第 4.3 节 - 改进测试过程[ISTQB®_ELTM_ITP_SYL]以获得 GQM 方法更完整的描述。）应当注意的是，GQM 方法并非总是与性能测试过程相适应。例如，某些度量代表着系统的健康状况，与性能目标没有直接的关联。

重要的是要意识到，在定义并捕获初始测量之后，可能需要进一步的测量和度量来理解真实的性能水平并确定可能需要采取纠正措施的位置。

2.2 汇总性能测试的结果

汇总性能度量的目的是为了以一种能够准确表达系统性能总体情况的方式来理解和表达它们。当仅在细节级别查看性能度量时，很难得出正确的结论- 尤其对于那些业务利益干系人而言。

对于许多利益干系人而言，其主要关注点是系统、网站或其他测试对象的响应时间在可接受的范围内。

一旦对性能度量有了更深入的理解，就可以汇总度量，以达到下列目标：

- 业务和项目的利益干系人可以看到系统性能的“大画面”–即整体状况
- 可以识别出性能趋势
- 可以用一种易理解的方式报告性能度量

2.3 性能度量的关键来源

系统性能应该最小限度地受度量收集工作(称为“探针效应”)的影响。此外，性能度量收集过程中对容量、精准度和速度的要求也促使工具使用成为必需。虽然组合使用工具并不少见，但这会在测试工具的使用中引入冗余及其它问题（参见第 4.4 节）。

性能度量有三个关键的来源：

性能测试工具
所有性能测试工具都会提供测量和度量作为测试结果。不同工具在度量数量、展示方式、以及针对特定场景为用户提供度量的定制化能力等方面，都会有所差异。（参见 5.1 节）

有些工具以文本格式收集并显示性能度量，而更为强大一些的工具则以图形化仪表盘的形式收集和显示性能度量。许多工具提供度量的导出功能，方便测试的评估和报告。

性能监控工具

性能监控工具通常作为性能测试工具报告能力的补充（另请参见第 5.1 节）。此外，监控工具可用于系统性能的持续性监控，并提醒系统管理员降低性能水平和提高系统错误和警报级别。这些工具还可用于在系统发生可疑行为时的探测和通知（例如拒绝服务攻击和分布式拒绝服务攻击）。

日志分析工具

这些工具可以扫描服务器日志并从中汇集度量。其中一些工具可以创建图表以提供数据的图形视图。

错误，警报和警告通常记录在服务器日志中。包括如下信息：

- 高资源使用率，如高 CPU 利用率、高磁盘存储量消耗，带宽不足
- 内存错误和警告，如内存耗尽
- 死锁和多线程问题，尤其是在执行数据库操作时
- 数据库错误，如 SQL 异常和 SQL 超时

2.4 性能测试的典型结果

在功能测试中，特别是在验证特定功能需求或用户故事的功能元素时，通常可以清楚地定义预期结果并解释测试结果以明确测试是否通过。例如，月度销售报表要么显示出正确的总额，要么显示不正确的总额。

虽然功能的适用性测试通常受益于明确定义的测试结果参照物，但性能测试常常缺乏这种信息来源。不仅利益干系人在阐明性能需求方面出了名的不好，许多业务分析师和产品负责人也都不善于引出这些需求。在定义预期的测试结果方面，测试人员能获得的指导通常很有限。

在评估性能测试结果时，对结果进行仔细审查是非常重要的。由于最初的原始结果可能具有误导性，失效会隐藏在明显良好的整体结果之下。例如，对于所有关键潜在瓶颈资源，资源利用率可能远低于 75%，但关键业务或用例的吞吐量或响应时间却慢上一个数量级。

要评估的具体结果依赖于正在执行的测试而有所不用，通常包括第 2.1 节中讨论的内容。

3. 软件生命周期中的性能测试– 195 分钟

关键词
度量、风险、软件开发生命周期、测试日志

学习目标

3.1 性能测试的主要活动
PTFL-3.1.1（K2）理解性能测试的主要活动

3.2 不同架构的性能风险
PTFL-3.2.1 (K2)解释不同架构的典型性能风险类别

3.3 软件生命周期中的性能风险
PTFL-3.3.1（K4）在软件开发生命周期中，分析给定产品的性能风险

3.4 性能测试活动
PTFL-3.4.1 (K4)分析给定的项目，决定在软件开发生命周期的每一个阶段恰当的性能测试活动

3.1 性能测试的主要活动

性能测试本质上是迭代的。每次测试都对应用和系统性能提供了有价值的洞察。从一次测试中收集到的信息，可用于修正或优化应用及系统参数。下一次测试迭代将显示修正的结果，以此类推，直到达到测试目标。

性能测试活动与 ISTQB®测试过程一致【ISTQB®_FL_SYL】

测试计划
由于需要分配测试环境、测试数据、工具和人力，所以性能测试的测试计划特别重要。此外，这也是确立性能测试范围的活动。

在测试计划过程中完成风险识别与风险分析活动，将相关信息更新到所有的测试计划文档中（如，测试计划，级别测试计划）。正如测试计划会根据需要被重新审视和修改一样，风险、风险级别和风险状态也需要修改来反映风险条件的变化。

测试监督与控制
针对已知的可能影响性能效率的问题定义控制措施、提供行动计划，如：

- 如果基础设施没有按特定的性能测试计划生成所需要的负载，则增加负载生成容量
- 硬件被变更，更新，或取代
- 网络组件发生变更
- 软件实现发生变革

评估性能测试目标来检查是否达到出口准则。

测试分析
有效的性能测试是基于对性能测试需求、测试目标、服务级别协议、IT 架构、过程模型以及其它测试依据的分析。此活动可以通过使用电子格式或容量计划工具，对系统资源需求和（或）行为进行建模和分析。
识别特殊测试条件，如负载级别、时间条件和待测事务等。之后决定需要的性能测试类型。

测试设计
进行性能测试用例的设计。这些通常以模块化形式构建，可能像拼插积木一样将它们用于更大、更复杂的性能测试中。（见 4.2 节）

测试实施
在测试实施阶段，性能测试用例被安排到性能测试规程中。这些性能测试规程应该反映用户正常执行步骤，和其它将被性能测试覆盖的功能性活动。

测试执行是在每次测试实施活动之前，建立和（或）重置测试环境。由于性能测试典型地由数据驱动，因此需要一个过程来创建在数据量及类型上代表实际生产的测试数据，以便可以模拟生产。

测试执行
测试执行发生在性能测试开始时，通常使用测试工具执行。评估测试结果来决定系统的性能测试是否满足需求和其它已阐述的目标。报告所有的缺陷。

测试结束
在测试总结报告中，把性能测试结果提交给利益干系人(如，架构师、经理、产品负责人)。结果通过已汇总的度量来展示，简化了测试结果含义的理解。可视化的报告，如仪表盘比基于文字的度量结果更容易理解。

性能测试通常被认为是一个多次的、在多个测试级别的、持续进行的活动（组件、集成、系统、系统集成和验收测试）。在规定的性能测试周期结束时，当设计的测试、测试工具资产（测试用例及测试规程）、测试数据或其它测试套件已完成，或已经交接给其它测试人员以便在系统维护活动中使用时，测试可能已经达到了测试结束活动的某个点。

3.2 不同架构的性能风险类型

如前所述，应用或系统性能因架构、应用和主机环境不同而变化。虽然不可能为所有系统提供一个完整的性能风险清单，但下面列表包括一些与特定架构相关的典型的风险类型：

单一计算机系统
这些系统或应用完全运行在一台非虚拟化计算机上。性能下降可能由于：
- 过多的资源消耗包括内存泄漏、后台活动，如安全软件、慢速存储子系统（如，低速外部设备或磁盘碎片），和操作系统管理不善。
- 不能充分利用可用资源（如，主存储器）的低效率算法，导致执行速度低于要求。

多层系统
这些系统是运行在多个服务器上的系统。每个服务器执行一组特定的任务，如数据库服务器、应用程序服务器和展示服务器。每台服务器当然都是一台计算机并受制于前面给出的风险。此外，由于糟糕的或不可伸缩的数据库设计、网络瓶颈以及任何单个服务器上带宽或容量不足性能可能会下降。

分布式系统
这些是综合系统，类似于多层系统架构，但是不同的服务器可以动态变化，如一个电子商务系统，可以根据下单子客户的不同地理位置访问不同的库存数据库。除了与多层架构相关的风险外，这些架构还可能由于关键工作流和数据流流自、流经、流向不稳定或不可预测的远程服务器而遭遇到性能问题，特别是这些服务器遭遇到周期性连接问题或间歇性高负载时。

虚拟系统
这些系统的物理硬件承载多个虚拟机。这些虚拟机可以承载单个计算机系统和应用程序，以及属于多层或分布式体系架构的服务器。典型的由虚拟引起的性能风险包括承载所有虚拟机的硬件负载过高，或主机虚拟机配置不当，导致资源不足。

动态或基于云系统
这些系统提供了按需伸缩的能力，当负载水平增加时增加容量。此系统是典型的分布式和虚拟化的多层系统，尽管有专门为减轻与这些系统架构相关的一些性能风险而设计的自伸缩功能，但仍然存在着在初始设置或后续更新时未被恰当配置的风险。

客户端服务器系统
这些系统运行在客户端上，该客户端通过用户界面与单个服务器、多层服务器或分布式服务器通信。由于有代码在客户端上运行，所以单个计算机风险适用于该代码，当然上面提到的服务器风险适用服务器端。此外，由于连接速度和可靠性问题、客户端连接点的网络拥塞（如公共 Wi-Fi）以及防火墙、数据包检查和服务器负载平衡等潜在问题，系统存在性能风险。

移动应用
这些是在智能手机、平板电脑或其它移动设备上运行的应用。这些应用程序会受到客户端服务器和基于浏览器（Web 应用程序）应用程序所提到的风险的影响。此外，移动设备上的有限的、变化的资源以及设备是否可连接（如受位置、电脑寿命、充电状态、设备上可用内存和温度），也会引起性能问题。对于使用设备传感器或无线电设备的应用，如加速器或蓝牙而言、来自它们的慢速数据流可能会造成问题。最后，移动应用通常与其它本地移动应用及远程 Web 服务器有大量的互动，这些都可能成为潜在的性能效率的瓶颈。

嵌入式实时系统
这些系统在日常活动中起作用，甚至控制着日常设备，如汽车（如，娱乐系统或智能制动系统）、电梯、交通信号、供暖、通风和空调（HVAC）系统等。这些系统通常有许多与移动设备相关的风险，包括日益增长的连接性问题，因为这些设备与互联网相连。尽管手机游戏的性能下降通常不会对用户构成安全危险，但是汽车制动系统的变慢可能会带来灾难性后果。

大型机应用程序
这些应用程序-在许多情况下已经有几十年的历史，通常支持数据中心的关键业务功能，有时也通过批量处理完成业务。当按照最初设计的方式使用时，大多数都是可预测的和快速的，但是其中许多现在可以通过 API、Web 服务或其数据库进行访问，这会导致意外的负载，从而影响已建立应用程序的吞吐量。

值得注意的是，任何特定的应用或系统可能应用到以上清单中的两个或两个以上的架构，这意味着所有相关风险将适用于那个应用程序或系统。实际上，考虑到物联网和爆炸式增长的移动应用--两个领域的交互和互联达到了极致--，所有的架构都可能以某种形式出现在一个应用程序中，那么所有的风险都是适用的。

当然架构显然是一个重要的技术决策，它对于性能风险的影响是深远的，其它的技术决策也会影响和制造风险。例如，内存泄漏更多出现在允许直接进行堆的内存管理的编程语言中，如 C 和 C++、并且关系数据库与非关系数据库的性能问题也是不同的。这些决策一直延伸到单个函数或方法的设计（例如，选择递归算法而不是迭代算法）。作为一名测试人员，根据他在一个组织和软件开发生命周期的角色和职责的不同，他可以了解甚至影响这些决策的能力也是不同的。

3.3 软件开发生命周期中的性能风险管理

在 ISTQB® 的多个大纲中，都涉及到了软件产品风险的过程（例如，参见[ISTQB®_FL_SYL]和[ISTQB®_ALTM_SYL]）。您可以从商业或技术角度分析（分别参见[ISTQB®_ALTA_SYL]和[ISTQB®_ALTTA_SYL]），以及特定质量属性相关的角度（如，[ISTQB®_UT_SYL]），找到特定风险及其考虑的相关讨论。在本节中，焦点是针对产品质量性能的相关风险，包括过程、参与者和考虑因素的变化。

对于产品质量性能相关风险，流程为：

1. 识别产品质量风险，专注于时间特征、资源使用及容量等特征。

2. 评估已经识别的风险，确保相关的体系结构类别被重视起来（参见第 3.2 节）。使用已经明确定义的标准，逐一评估已经识别的风险其可能性、严重程度和风险级别。
3. 针对每项风险的本质和风险级别，采取恰当的风险缓解措施。
4. 持续管理风险，确保风险在发布前得到充分的缓解。

通常质量风险分析需要同时有业务及技术利益干系人的参与。对于与性能相关的风险分析，业务利益干系人，特别是那些具备认识产品性能问题在实践中如何影响客户、用户和业务，以及其他下游利益干系人能力的人必须参与。业务利益干系人必须认识到预期的使用、业务、社会或安全临界、潜在的财务和/或声誉损害、民事或刑事法律责任以及类似的因素，会从业务角度影响风险、生成风险或影响失效的严重程度。

此外，技术利益干系人必须包括那些对影响性能的相关需求、架构、设计和实现决策有深刻理解的人。技术利益干系人必须从技术的角度认识到架构、设计和实现决策对性能风险的影响，生成风险并影响缺陷发生的可能性。

所选择的特定风险分析过程应该具有适当的正式性和严谨性。对于与性能相关的风险，尤其重要的风险应尽早开始风险分析过程，并定期重复。换句话说，测试人员应该避免完全依赖于在系统测试级别和系统集成测试级别结束时进行的性能测试。许多项目，特别是更大、更复杂系统的系统项目，由于在项目早期的需求、设计、体系结构和实现决策所引入的性能缺陷到项目晚期才被发现，而遇到了不幸的意外。因此重点应该放在贯穿软件开发生命周期的性能风险识别、评估、缓解和管理的迭代方法上。

例如，如果大批量的数据由关系数据库处理，那么由于糟糕的数据库设计导致的多对多连接的缓慢性能可能只会在使用大型测试数据集(例如，系统测试期间使用的数据集)进行动态测试时显现出来。然而，开展一次有经验的数据库工程师参与的技术审查，可以在数据库实现之前预测问题。在这样的评审之后，可在迭代方法中再次确定和评估风险。

此外，风险降低和管理必须贯穿并影响整个软件开发过程，不仅仅是在动态测试中。例如，当关键性性能相关的决策，如，期望事务数量或并发用户不能在项目早期确认，重要的是设计和架构决策允许高度可伸缩性（如，随需应变的基于云计算的资源）。这可以使早期的风险缓解决策成为可能。

好的性能工程师可以帮助项目团队避免在较高的测试级别(例如系统集成测试或用户验收测试)中发现关键的性能缺陷。在项目的后期发现的性能缺陷可能非常昂贵，甚至可能导致整个项目的取消。

与任何类型的质量风险一样，与性能相关的风险永远无法完全避免，即，与性能相关的生产失败风险将始终存在。因此，风险管理流程必须包括向流程中涉及的业务和技术利益相关人提供具体而现实的剩余风险级别评估。例如，简单地说“是的，客户在退房期间仍然有可能经历长时间的延迟”是没有帮助的，因为它不知道已经有多少风险被缓解，也不知道仍然存在的风险水平。相反，提供清晰的信息，了解有多少客户可能遇到至少某个阈值的延迟，将有助于人们了解当前的状态。

3.4 性能测试活动

性能测试活动将根据所使用的软件开发生命周期的类型，以不同的方式组织和执行。

顺序开发模型

在顺序开发模型中进行性能测试的理想实践是将性能标准包括在项目开始时定义的验收标准中。加强测试的生命周期视图，性能测试活动应该在整个软件开发生命周期中进行。随着项目的进展，每个连续的性能测试活动都应该基于前面活动中定义的项目，如下所示：

- 概念-验证系统性能目标被定义为项目的验收标准。
- 需求-验证性已定义的性能需求，并正确地表示利益相关人需求。
- 分析和设计-验证系统设计是否反映了性能需求。
- 编码/实现-验证代码的有效性，并在需求和设计中反映性能。

- 组件测试-进行组件级性能测试。
- 组件集成测试-在组件集成级别进行性能测试。
- 系统测试-在系统级进行性能测试，包括能代表生产环境的硬件、软件、程序和数据。可以模拟系统接口，只要它们能真实地反映性能。
- 系统集成测试-在代表生产环境的整个系统中执行性能测试。
- 验收测试-验证系统性能满足最初声明的用户需求和验收标准。

迭代和增量开发模型

在这些开发模型中，如敏捷，性能测试也被看作是迭代的和增量的活动(参见[ISTQB®_FL_AT])。性能测试可以作为第一个迭代的一部分进行，也可以作为一个完全用于性能测试的迭代进行。然而，使用这些生命周期模型，性能测试的执行可以由一个单独的团队执行，该团队的任务是性能测试。

持续集成(CI)通常在迭代和增量软件开发生命周期中执行，这有助于测试的高度自动化执行。在 CI 中最常见的测试目标是执行回归测试，并确保每个构建都是稳定的。性能测试可以是 CI 中执行的自动化测试的一部分，如果测试的设计方式是在构建级别上执行的。然而，与功能的自动化测试不同的是，还有其他一些关注点，如：

- 性能测试环境的设置 - 这通常需要随需应变的测试环境，例如基于云的性能测试环境。
- 决定在 CI 中自动化哪些性能测试-由于 CI 测试可用的时间较短，CI 性能测试可能是由专家团队在迭代期间的其他时间执行的更广泛的性能测试的子集。
- 为 CI 创建性能测试- 作为 CI 的一部分，性能测试的主要目标是确保变更不会对性能产生负面影响。根据在任何给定构建上做的更改，都需要进行新的性能测试。
- 对应用程序或系统的某些部分执行性能测试-这通常要求工具和测试环境能够进行快速性能测试，包括能够选择适用测试子集。

在迭代和增量软件开发生命周期中的性能测试也可以有自己的生命周期活动：

- 发布计划–在一个发布中，从第一个迭代到最后一个迭代，所有的迭代和增量软件开发生命周期中都要考虑性能测试。确定和评估性能风险，并计划缓解措施。这通常包括在发布应用程序之前，计划最终的性能测试。
- 迭代计划–在线一次迭代的上下文中，性能测试可以在迭代中执行，伴随每次迭代完成。对每个用户故事中的性能风险进行更详细的评估。
- 用户故事创建–在敏捷方法中，用户故事通常是性能需求的基础，特定的已描述的性能标准与验收标准相关。这些通常被称为“非功能性”用户故事。
- 设计性能测试– 被描述在特性的用户故事中的性能需求和标准，可用于测试的设计。
- 编码/实现– 在编码过程中，可以在执行性能测试。例如，优化算法以获得最优性能效率。
- 测试/评估– 通常测试围绕着开发活动执行，但是基于在迭代周期内性能测试的范围和目标，性能测试也可以作为单独的活动执行。例如，如果性能测试的目标是：将迭代的性能作为一组完整的用户故事进行测试，那么需要比单个用户故事的性能测试范围更广的性能测试。这可以安排在一个专门的迭代中进行性能测试。
- 交付– 由于交付将应用程序引入生产环境，因此需要监控性能来确定应用程序在实际中是否达到了预期的性能水平。

商用现货软件（COTS）和其它供应商或收购模式

许多组织自身并不开发软件和系统，但是从供应商采购软件，或开源项目中获得软件。在这种供应商或收购模型中，性能是一个重要的模型，需要同时从供应商（供应商/开发人员）和收购方（客户）角度进行测试。

不论应用的来源是什么，客户都有责任验证性能是否满足他们的需求。客户定制化的情况下，性能需求和与之相关的性能标准，应该包含在供应商与客户的合同中。在商用现货软件的模式下，客户仅有的责任是在部署之前，在现实的测试环境中测试产品的性能。

4. 性能测试任务– 475 分钟

关键词

并发性、负载配置文件、负载生成、操作配置文件、缓慢下降（ramp down）、逐渐增加（ramp up）、综合系统、系统吞吐量、测试计划、思考时间、虚拟用户

学习目标

4.1 策划
PTFL-4.1.1（K4）从相关信息中得出性能测试目标
PTFL-4.1.2（K4）考虑给定项目的性能指标生成性能测试计划纲要
PTFL-4.1.3（K4）创建一个演示文稿，使不同干系人能够理解计划性能测试背后的基本原理。

4.2 分析、设计和实施
PTFL-4.2.1（k2）给出了性能测试中遇到的典型协议示例
PTFL-4.2.2（k2）了解性能测试中事务的概念
PTFL-4.2.3（k4）分析系统使用的操作配置文件
PTFL-4.2.4（k4）根据给定性能目标的操作配置文件创建派生的负载配置文件
PTFL-4.2.5（K4）在开发性能测试时分析吞吐量和并发性
PTFL-4.2.6（k2）了解性能测试脚本的基本结构
PTFL-4.2.7（k3）执行与计划以及负载配置文件一致的性能测试脚本
PTFL-4.2.8（k2）了解准备性能测试执行所涉及的活动。

4.3 执行
PTFL-4.3.1（k2）了解运行性能测试脚本的主要活动

4.4 分析结果和报告
PTFL-4.4.1（K4）分析并报告性能测试结果和影响

4.1 策划

4.1.1 推导性能测试目标

干系人可能包括用户和具有业务或技术背景的人员。他们可能对性能测试有不同的目标。干系人制定目标，确定使用的术语，并制定判断目标是否实现的准则。

性能测试的目标与这些不同类型的干系人相关。区分基于用户的目标和技术目标是一个很好的实践。基于用户的目标主要集中在最终用户满意度和业务目标上。通常，用户不太关心功能类型或产品是如何交付的。他们只是想做他们需要做的事情。

另一方面，技术目标侧重于操作方面，回答与系统扩展能力相关的问题，或在什么情况下性能下降可能变得明显。

性能测试的关键目标包括识别潜在风险、寻找改进机会和识别必要的变更。
从不同干系人处收集信息时，应回答以下问题：

- 性能测试中将执行哪些事务，预期的平均响应时间是多少？
- 要捕获哪些系统度量（例如内存使用率、网络吞吐量）以及预期值？
- 与之前的测试周期相比，通过这些测试我们期待得到哪些性能改进？

4.1.2 性能测试计划

性能测试计划（PTP）是在进行任何性能测试之前创建的文档。测试计划应参考 PTP（见[ISTQB®_FL_SYL]），该计划还包括相关的调度信息。一旦性能测试开始，它就会持续更新。

PTP 中应提供以下信息：

目标
PTP 目标描述了性能测试的目标、策略和方法。它可以量化地回答系统在负载下运行的充分性和完备性这一核心问题。

测试目标
针对每种类型的干系人，列出了待测试系统（SUT）要实现的性能效率的总体测试目标（见第 4.1.1 节）。

系统概述
SUT 的简要描述将为性能测试参数的测量提供背景。概述应包括对负载下被测功能的简要描述。

要进行的性能测试类型
列出将要进行的性能试验的类型清单（见第 1.2 节）并说明每种类型的目的。

验收标准
性能测试旨在确定在给定的工作负载下系统的响应特性、吞吐量、可靠性和/或可扩展性。一般来说，响应时间是用户关注的问题，吞吐量是业务关注的问题，资源利用是系统关注的问题。应为所有相关测量制定验收标准，并与以下内容（如果有的话）相关：

- 总体性能测试目标
- 服务水平协议（SLA）
- 基线值 - 基线是一组度量，用于比较当前和以往的性能指标。这可以证明特定的性能改进和/或确认测试验收标准是否实现。首次创建基线可能需要使用来自数据库的净化数据。

测试数据
测试数据包括一系列需要为性能测试定义的数据。这些数据可以包括以下内容：

- 用户帐户数据（例如，可同时登录的用户帐户）
- 用户输入数据（例如，用户为了执行业务流程而进入应用程序的数据）
- 数据库（例如，预填充数据库，测试中使用的数据事先填充到数据库）

测试数据创建过程应考虑以下几个方面：

- 从生产数据中提取数据
- 将数据导入 SUT
- 创建新数据
- 创建备份，以便在执行新的测试周期时恢复数据
- 数据屏蔽或隐匿。该实践用于含有可识别出个人信息的生产数据，在通用数据保护条例（GDPR）下数据屏蔽或隐匿是强制性的。然而，在性能测试中，数据屏蔽增加了性能测试的风险，因为它可能不具有与真实世界相同的数据特性。

系统配置

PTP 的系统配置部分包括以下技术信息：

- 特定系统架构的描述，包括服务器（如 Web、数据库、负载均衡器）
- 多层定义
- 计算硬件（如 CPU 核心、RAM、固态磁盘（SSD）、硬盘驱动器磁盘（HDD））的具体细节，包括版本
- 软件的具体细节（例如，应用程序、操作系统、数据库、用于支持企业的服务），包括版本
- 与 SUT 一起运行的外部系统及其配置和版本（例如，与 NetSuite 集成的电子商务系统）
- SUT 构建/版本标识符

测试环境

测试环境通常是模拟生产的独立环境，但规模较小。PTP 的这一部分应包括如何推算出性能测试的结果，以适用于更大的生产环境。对于某些系统，生产环境成为测试的唯一可行选项，但在这种情况下，必须讨论这种类型测试的特定风险。

测试工具有时位于测试环境本身之外，可能需要特殊的访问权限才能与系统组件交互。对于测试环境和配置来说需要有所考虑。

性能测试也可使用系统的一个部件进行，该部件能够在没有其他部件的情况下运行。这通常比使用整个系统进行测试便宜，并且可以在开发组件后立即执行。

测试工具

本节描述了哪些测试工具（和版本）将用于编写脚本、执行和监控性能测试（请参见第 5 章）。此列表通常包括：

- 用于模拟用户事务的工具
- 在系统架构（architecture）中从多个点生成负载的工具（存在点）
- 监控系统性能的工具，包括系统配置下的上述工具

配置文件

操作配置文件为系统的特定用途提供了一个可重复的步骤顺序。综合这些操作配置文件会产生一个负载配置文件（通常称为场景）。有关配置文件的更多信息，请参见第 4.2.3 节。

相关度量

在性能测试执行期间，可以收集大量的测量和度量（参见第 2 章）。但是，采取过多的度量可能会使分析变得困难，并对应用程序的实际性能产生负面影响。出于这些原因，识别与实现性能测试目标最相关的测量和度量是很重要的。

下表（在第 4.4 节中更详细地解释）显示了一组典型的性能测试和监控度量。在需要时，应为项目定义性能测试目标：

性能度量	
类型	**度量**
虚拟用户状态	# Passed 通过 # Failed 失败
事务响应时间	Minimum 最短时间 Maximum 最长时间 Average平均时间 90% Percentile 90%响应时间
每秒事务数	# Passed / second 通过数/秒 # Failed / second失败数/秒 # Total / second总数/秒
点击率（e.g., on database or web server 例如,在数据库或者 web服务器上）	Hits / second 点击量/秒 ▪ Minimum 最小点击率 ▪ Maximum 最大点击率 ▪ Average 平均点击率 ▪ Total总点击率
吞吐量	Bits / second 比特/秒 ▪ Minimum 最小比特率 ▪ Maximum 最大比特率 ▪ Average 平均比特率 ▪ Total总比特率
HTTP每秒响应数 Responses Per	Responses / second 响应数/秒 ▪ Minimum 最小响应数 ▪ Maximum最大响应数 ▪ Average平均响应数 ▪ Total 总响应数 Response by HTTP/HTTP协议响应数 response codes代码响应数

性能监控	
类型	**度量**
CPU 使用概况	% CPU 占用量率
Memory 内存使用概况	% 内存占用率

风险
风险可能包括部分未做性能测试的区域以及性能测试的局限（例如，无法模拟的外部接口、负载不足、无法监控服务器）。测试环境的局限性也可能产生风险（例如，数据不足、环境缩小）。更多风险类型见第 3.2 和 3.3 节。

4.1.3 关于性能测试的沟通

测试人员必须能够向所有利益干系人传达性能测试方法以及要开展的活动背后的基本原理（如性能测试计划中所详述）。在这种沟通中要讨论的主题可能因利益干系人而有很大差异，这取决于他们是对“业务/面向用户”，还是对“技术/面向操作”更关注。

关注业务的利益干系人
在与关注业务的利益干系人沟通时，须考虑下列因素：
- 关注业务的利益干系人不太关注于功能性和非功能性质量特征之间的区别。
- 有关工具、脚本与负载生成的技术问题一般是次要的。
- 必须清晰表述产品风险与性能测试目标之间的关联。
- 必须让利益干系人了解，与实际生产条件相比，性能测试结果具有多大程度的代表性，与计划的测试成本有关。在两者之间有一个平衡。
- 必须传达计划的性能测试的可重复性：测试是难以重复，还是能够较为容易的重复呢？
- 必须传达项目风险，包括在测试设置、基础设施需求（例如硬件、工具、数据、带宽、测试环境、资源）方面的约束与依赖条件以及对关键员工的依赖性。
- 必须传达高级别的活动（请见章节 4.2 与 4.3），以及包含成本、时间进度表和里程碑的整体计划。

关注技术的利益干系人
在与关注技术的利益干系人沟通时，须考虑下列因素：
- 必须阐明生成所需负载配置文件的计划方法，并明确技术利益干系人的预期参与。
- 必须阐明性能测试设置和执行中的详细步骤，以显示测试与架构风险的关系。
- 必须传达性能测试重复所需的步骤，包括组织方面（例如关键员工的参与）以及技术问题。
- 测试环境为共用的情况下，必须沟通好性能测试的时间安排，以确保测试结果不会受到不利影响。
- 如性能测试需在生产环境执行，必须沟通和落实如何降低对实际用户的潜在影响。
- 技术利益干系人必须清楚他们的任务以及时间表。

4.2 分析、设计与实施

4.2.1 典型通信协议

通信协议定义了一组计算与系统间的通信规则。针对系统特定部分进行正确的测试设计，需要理解这些协议。

通信协议通常由开放式系统互连（OSI）模型层（参见 ISO / IEC 7498-1）描述，不过某些协议可能不属于该模型。对于性能测试，从第 5 层（会话层）到第 7 层（应用层）的协议最常用于性能测试。常用协议包括：

- 数据库- ODBC, JDBC，其他供应商特定协议
- Web - HTTP, HTTPS,HTML
- Web 服务 - SOAP,REST

一般而言，性能测试中最关注的 OSI 层级与被测试的架构层级有关。例如，在测试某些低层级，如嵌入式架构时，OSI 模型中较低的层级将是测试重点。

性能测试中使用的其他协议包括：

- 网络- DNS, FTP, IMAP, LDAP, POP3, SMTP, Windows Sockets,CORBA
- 移动 - TruClient, SMP,MMS
- 远程访问 - Citrix ICA,RTE
- SOA - MQSeries, JSON,WSCL

性能测试可以在单个系统组件（例如 Web 服务器、数据库服务器）上或通过端到端测试在整个系统上执行，因此理解总体系统架构非常重要。运用主从式模型构建的传统 2 层应用程序将“客户端”指定为 GUI 和主用户界面，将“服务器”指定为后端数据库。这些应用程序需要使用 ODBC 等协议来访问数据库。随着基于 Web 的应用程序和多层体系结构的发展，许多服务器都参与处理最终呈现给用户浏览器的信息。

根据要测试的系统部分，需要理解该部分所使用的相应协议。因此，如果需要执行模拟用户在浏览器上活动的端到端测试，则将使用诸如 HTTP/HTTPS 的 web 协议。通过这种方式，可避开与图形用户界面的交互，测试可以关注后端服务器的通信和活动。

4.2.2 事务处理

事务描述了系统从启动点到完成一个或多个流程（请求、操作或操作流程）时执行的一组活动。可通过测量事务的响应时间来评估系统性能。在性能测试期间，这些测量用于识别需要校正或优化的任何组件。

模拟事务可以包括思考时间以更好地反映真实用户活动的时间（例如按下“发送”按钮）。事务响应时间加上思考时间等于该事务的已用时间。

在性能测试期间收集的事务响应时间展示了它在系统不同负载下的变化情况。分析可能显示有些结果在负载下没有恶化，而有些结果严重恶化。通过逐渐增加负载和测量底层事务时间，可以将恶化的原因与一个或多个事务的响应时间关联起来。

事务也可以嵌套，这样可以测量单个和多个活动。例如，在测试在线订购系统的性能效率时，这可能会有所帮助。测试人员可能想要测量订单处理中的各个步骤（例如搜索物品、将物品添加到购物车、支付物品、确认订单）以及整个订单处理。通过嵌套事务，可以在一次测试中收集这两组信息。

4.2.3 识别操作配置文件

操作配置文件明确规定了与应用程序（例如来自用户或其他系统组件）的不同交互模式。既定的应用程序可指定多个操作配置文件，它们可以组合在一起，创建所需的负载配置文件，以实现特定的性能测试目标（参见第 4.2.4 节）。

本节描述用于识别操作配置文件的主要步骤，具体如下：

1. 确定要收集的数据
2. 使用一个或多个来源收集数据
3. 评估数据以构建操作配置文件

识别数据

当用户与被测系统交互时，收集或估计以下数据以便对其操作配置文件进行建模（即他们如何与系统交互）：

- 不同类型的用户类型及其角色（例如：标准用户、注册会员、管理员、具有特定权限的用户组）。
- 由这些用户/角色执行的、不同的常规任务（例如：浏览网站以获取信息，在网站中搜索特定产品，执行特定于角色的活动）。请注意，这些任务通常在高抽象级别（例如，在业务过程或主要用户故事的级别）进行最佳建模。
- 在给定时间段内每单位时间内每个角色/任务的预估用户数。此信息也有利于随后构建负载配置文件（参见第 4.2.4 节）。

采集数据

上面提到的数据可以从许多不同的来源采集：

- 与干系人进行访谈或研讨会，例如产品所有者、销售经理和（潜在的）最终用户。这些讨论往往能展示出用户的主要操作配置文件，并给出基本问题“谁是此应用程序的目标用户”的答案。
- 功能规范和需求（如有的话）是有关预期使用模式的有价值的信息来源，这些信息有助于识别用户类型及其操作配置文件。在功能规范以用户故事来表达的情况下，标准格式直接允许识别用户类型（即：作为一种<用户类型>，我想要<某个功能>以达到<某些成效>）。类似地，UML 用例图和描述标识用例的“参与者”。
- 评估从类似应用程序获得的使用数据和度量有助于识别用户类型，并提供预期用户数量的初步指示。建议访问自动监控的数据（例如，从网站管理员的管理工具），这将包括监控日志和从当前操作系统使用中获取的数据，且该系统已计划进行更新。
- 观察用户在使用应用程序执行预定义任务时的行为可以深入了解为性能测试建模的操作配置文件类型。建议将此任务与任何有计划的易用性测试进行统筹安排（特别是在易用性实验室可用的情况下）。

构建操作配置文件

以下步骤用于识别和构建用户的操作配置文件：

- 采用自上而下的方法。最初创建的是相对简单的广泛的操作概要，只有在需要实现性能测试目标时才能进一步分解（见第 4.1.1 节）。
- 如果特定用户配置文件涉及频繁执行的任务、需要在不同系统组件之间进行关键（高风险）或频繁的事务处理，或者可能需要传输大量数据，则可以将其单独列出，作为性能测试的相关内容。
- 在用于创建负载配置文件之前，与主要利益干系人一起审查和完善操作配置文件（参见第 4.2.4 节）。

被测系统并非始终承受用户施加的负载。以下类型系统的性能测试也可能需要操作配置文件（请注意包括但不限于）。

离线批处理系统

此处重点主要在于批处理系统的吞吐量（参见第 4.2.5 节）及其在指定时间段内完成的能力。操作配置文件侧重于批处理所需的处理类型。例如，股票交易系统的操作配置文件（通常包括在线和基于批量的交易处理）可以包括那些和支付交易有关的，验证凭证有关的，检查特定类型股票交易法规一致性有关的操作配置文件。这些操作配置文件中的每一个都将导致在股票交易的整个批处理过程中采用不同的路径。上面概括的用于识别基于用户的在线系统的操作配置文件的几个步骤也可以应用于批处理上下文中。

综合系统
多系统（嵌入式亦可）环境中的组件响应来自其他系统或组件的不同类型的输入。根据被测系统的性质，这可能需要对几个不同的操作配置文件进行建模，以有效地表示这些供应商系统提供的输入类型。这可能涉及与系统架构师一起、且基于系统和接口规范的详细分析（例如，缓冲器和队列）。

4.2.4 创建负载配置文件

负载配置文件指定了被测组件或系统在生产环境中可能经历的活动。它由指定数量的实例组成，这些实例将在指定时间段内执行预定义的操作配置文件中的动作。在实例是用户的情况下，通常应用术语“虚拟用户”。

创建符合实际的且可重用的负载配置文件所需的主要信息是：

- 性能测试目标（例如，评估压力负载下的系统行为）
- 准确表示单个使用模式的操作配置文件（参见第 4.2.3 节）
- 已知的吞吐量和并发性问题（参见第 4.2.5 节）
- 操作配置文件将要被执行的数量和时间分布，从而使得 SUT 经历期望的负载。典型的例子是:
  - 逐渐增加：稳定增加负载（例如，每分钟添加一个虚拟用户））
  - 缓慢下降：稳定减少负载
  - 阶跃：负载的瞬时变化（例如，每五分钟增加100个虚拟用户）
  - 预定义的分布（例如，模仿日常或季节性商业周期的量级）

以下示例显示了以生成压力条件（等于或高于系统能处理的预期最大值）为目标的负载配置文件的构造。

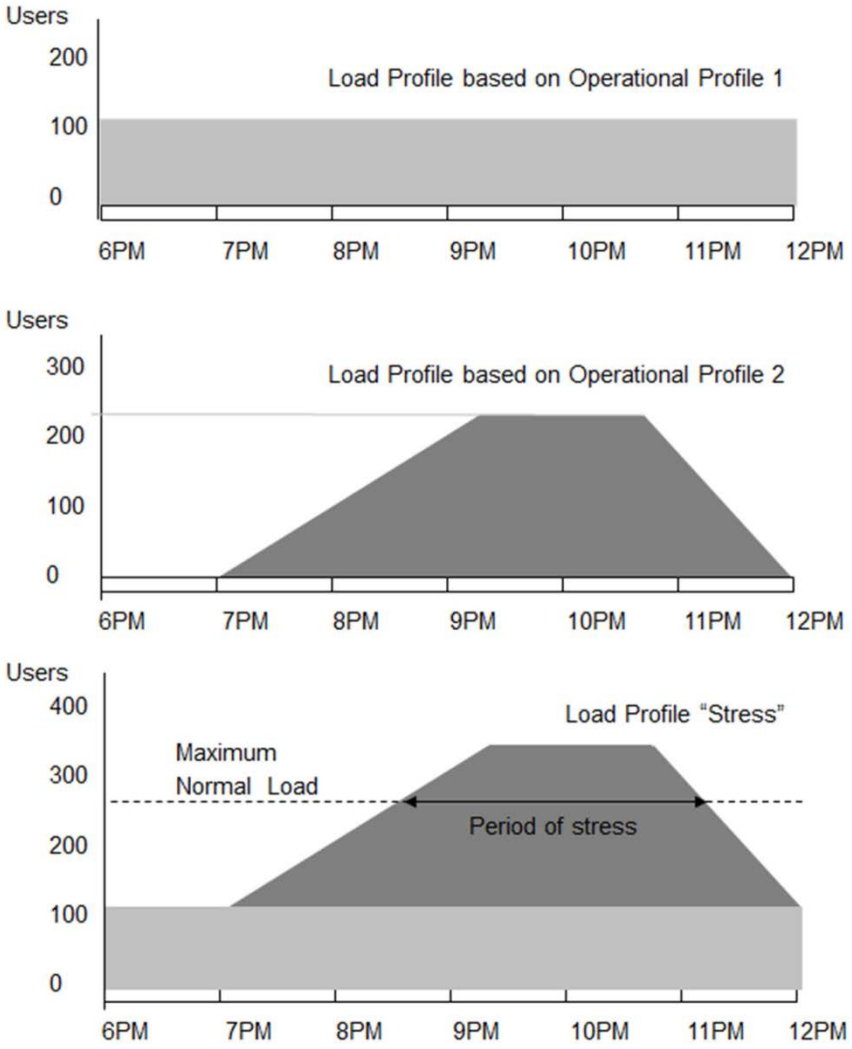

图 1：构造“压力”负载配置文件的示例

在图的顶部显示了一个负载配置文件，其中包含 100 个虚拟用户的阶跃输入。这些用户在整个测试期间执行操作配置文件 1 定义的活动。这在代表背景负载的许多性能负载配置文件中很典型。

中间的图表显示了一个负载配置文件，该配置文件包含一个逐渐增加到 220 个虚拟用户，这些用户在缓慢下降前保持两个小时。每个虚拟用户执行操作配置文件 2 中定义的活动。

最下面的图显示了由上述两者组合产生的负载配置文件。被测系统承受周期为 3 小时的压力。更多示例，请参阅 [Bath14]。

4.2.5 分析吞吐量和并发性

了解工作负载的不同方面非常重要：吞吐量和并发性。要正确建模操作和负载配置文件，这两个方面都必须考虑。

系统吞吐量

系统吞吐量是系统在单位时间内处理的给定类型的事务数量的测量。例如，每小时的订单数或每秒的 HTTP 请求数。系统吞吐量应该与网络吞吐量区分开来，网络吞吐量是通过网络传输的数据量（第 2.1 节）。

系统吞吐量定义了系统上的负载。不幸的是，通常并发用户的数量被用于定义交互式系统的负载，不是吞吐量。这种做法局部正确，因为并发用户数通常更容易得到，局部是因为负载测试工具定义负载的方式也通常如此。如果没有定义操作配置文件，每个用户正在做什么以及有多频繁（这也是一个用户的吞吐量），那么用户数量不是测量负载的好方法。例如，如果每分钟有 500 个用户运行短暂的查询操作，那么我们每小时的吞吐量为 30,000 次查询。如果相同的 500 个用户运行相同的查询，但每个用户每小时只查询一次，则吞吐量为每小时 500 次查询。因此，同样是 500 个用户，但负载之间相差 60 倍，并且系统的硬件需求至少也有 60 倍的差异。

工作负载建模通常通过考虑虚拟用户（执行线程）的数量和用户思考时间（用户操作与操作之间的延迟）来完成。但是，系统吞吐量也由处理时间定义，并且该时间可能随着负载的增加而增加。

系统吞吐量=[虚拟用户数]/（[处理时间]+[用户思考时间]）

因此，当处理时间增加时，即使其他所有变量保持不变，吞吐量也可能显著降低。

系统吞吐量是测试批处理系统时的一个重要方面。在这种情况下，通常根据在给定时间范围内可以完成的事务的数量（例如，夜间批处理窗口）来测量吞吐量。

并发性

并发性是测量同时/并行执行线程数的指标。对于交互式系统，它可能是若干同时/并行用户。并发性通常通过设置虚拟用户的数量在负载测试工具中建模。

并发性是一项重要的测量。它表示并行会话的数量，每个会话可以使用自己的资源。即使吞吐量相同，所使用的资源量也可能因并发性而异。典型的测试设置是封闭系统（从排队论的角度来看），其中设置了系统中的用户数（固定人口）。如果所有用户都在等待系统在封闭系统中的响应，则没有新用户可以到达。许多公共系统都是开放系统，即使所有当前用户都在等待系统的响应，新用户也会一直到达。

4.2.6 性能测试脚本的基本结构

性能测试脚本应该模拟向被测系统（可能是整个系统或其中一个组件）提供负载的一个用户或组件活动。它以适当的顺序以给定的速度向服务器发起请求。

创建性能测试脚本的最佳方法取决于使用的负载生成方法（第 4.1 节）。

- 传统方法是在协议级别录制客户端与系统或组件之间的通信，然后在脚本参数化和文档化后进行回放。参数化可以产生可扩展和可维护的脚本，但参数化任务可能是耗时的。
- 在 GUI 级别进行录制通常涉及使用测试执行工具捕获单个客户的 GUI 操作，并使用负载生成工具运行该脚本以代表多个客户。
- 可以使用协议请求（例如，HTTP 请求），GUI 操作或 API 调用来完成编程。在编写脚本的情况下，必须确定发送到实际系统和从实际系统接收的请求的确切顺序，这可能是很重要的。

通常，脚本是一个或多个代码段（用一般的编程语言编写，带有一些扩展名或使用特定语言）或对象，可以由带有界面（GUI）的工具呈现给用户。在这两种情况下，脚本将包括创建负载的服务器请求（例如，HTTP 请求）以及围绕它们的一些编程逻辑来指定如何调用这些请求（例如，以什么顺序，在什么时刻，使用什么参数，应该校验什么）。逻辑越复杂，使用强大的编程语言就越必要。

整体结构

通常，脚本具有初始化部分（为测试脚本主体准备好一切），可以被执行多次的主体部分以及清理部分（采取必要的步骤以正确结束测试）。

数据采集

为了收集响应时间，应将计时器添加到脚本中以测量请求或请求组合所花费的时间。定时请求应与有意义的逻辑工作单元匹配，例如，用于将商品添加到订单或提交订单的业务事务。

理解被测量的到底是什么非常重要：在协议级脚本的情况下，它只是服务器和网络响应时间，而 GUI 脚本测量的是端到端时间（尽管精确的测试内容取决于使用的技术）。

结果验证和错误处理

脚本的一个重要组成部分是结果验证和错误处理。即使在最好的负载测试工具中，默认的错误处理机制往往也很少（例如检查 HTTP 请求返回代码），因此建议添加额外的检查以验证请求实际返回的内容。此外，如果在出现错误时需要进行任何清理工作，则可能需要手动实施这些操作。一个不错的实践是使用间接方法来验证脚本是否在执行它应该执行的操作，例如，检查数据库以验证是否添加了正确的信息。

脚本可以包括其他逻辑，这些逻辑可以指定有关何时以及如何引发服务器请求的规则。一个示例是设置同步点，这是通过指定脚本在继续执行之前应该等待该同步点的一个事件发生来完成的。同步点可用于确保同时调用特定操作或协调多个脚本之间的工作。

性能测试脚本是软件，因此创建性能测试脚本是一项软件开发活动。质量保证和测试应包含在该过程中，以验证脚本是否可以使用整个输入范围的数据按预期工作。

4.2.7 实现性能测试脚本

性能测试脚本基于 PTP 和负载配置文件来实现。虽然实施的技术细节将根据使用的方法和工具而有所不同，但整体过程保持不变。使用集成开发环境（IDE）或脚本编辑器创建性能脚本，以模拟用户或组件行为。通常创建脚本以模拟特定的操作配置文件（尽管通常可以将一个脚本中的多个操作配置文件与条件语句组合在一起）。

当请求序列确定了，可以根据所选方法录制或编写脚本。录制通常保证它准确模拟真实的系统，而编写脚本依赖于对正确请求序列的了解。

如果使用协议级别的录制，则大多数情况下录制后的一个重要步骤是替换定义上下文的所有录制的内部标识符。必须将这些标识符转换为变量，变量可以在不同的脚本执行中被替换为从请求响应中提取出来的适当值（例如，在登录时获取并且在所有后续事务必须提供的用户标识符）。这是脚本参数化的一部分，有时也称为“关联”。在该上下文中，单词“关联”具有与在统计中使用时不同的含义（代表两个或多个事物之间的关系）。高级负载测试工具可以自动进行一些关联，因此在某些情况下可能是透明的，但在更复杂的情况下，可能需要手动关联或添加新的关联规则。不正确的相关性或缺乏相关性是录制的脚本无法重放的主要原因。

使用相同的用户名运行多个虚拟用户并访问相同的数据集（通常在没有做任何进一步必要的相关性修改的前提下回放录制的脚本时出现）很容易得到误导性结果。这是因为数据可以完全缓存起来（从磁盘复制到内存以便更快地访问），这样得到的结果将比生产环境中更好（生产环境可能从磁盘读取此类数据）。使用相同的用户和/或数据也可能导致并发问题（例如，如果数据在用户更新时被锁定）并且结果将比生产环境中的情况更糟糕，因为软件会在下一个用户可以锁定数据以进行更新之前等待解锁。

因此，脚本和测试框架应该被参数化（例如，固定或录制的数据应该被包含合理选项的列表中的值替换），以便每个虚拟用户使用适当的数据集。这里的术语“合理”意味着足够的差异性以避免系统，数据和测试需求所特有的缓存和并发带来的问题。这种进一步的参数化取决于系统中的数据以及系统使用该数据的方式，因此尽管许多工具能提供此类支持，该过程仍通常手动完成。

在某些情况下，必须对某些数据进行参数化，以使测试能够运行多次。例如，创建订单时，订单名称必须是唯一的。除非订单的名称已参数化，否则一旦尝试创建现有（录制的）名称的订单，测试将失败。

为匹配操作配置文件，应插入和/或调整用户思考时间（如果是录制的）以生成适当数量的请求/吞吐量，如第 4.2.5 节中所述。

当创建用于单独操作配置文件的脚本时，它们将组合成实现整个负载配置文件的方案。负载配置文件控制使用每个脚本启动的虚拟用户数，时间和参数。确切的实现细节取决于特定的负载测试工具或框架。

4.2.8 准备执行性能测试

准备执行性能测试的主要活动包括

- 配置被测系统
- 部署环境
- 配置负载发生器和监控工具，并确保监控工具能收集所有必要的信息。

保证测试环境尽可能接近生产环境是非常重要的，如果做不到，那么必须清楚了解环境差异以及预测测试环境的测试结果在生产环境中会如何变化。理想中是使用生产环境和数据，但在缩减版的测试环境中进行测试也可以规避一些性能风险。

必须记住，性能是环境的非线性函数，因此环境离生产标准越远，就越难对产品性能做出相应的预测。测试系统越不像产品，预测的不可靠性和风险系数也会相应增加。

测试环境最重要的部分是数据、硬件和软件配置以及网络配置。数据的大小和结构对负载测试结果有很大的影响。，使用一小组数据样本集或具有不同数据复杂性的样本集进行性能测试可能会产生误导结果，特别是当生产系统将使用大量数据集时。在执行实际测试之前，很难预测数据大小对性能的影响有多大。测试数据与生产数据在数据大小和结构上越接近，测试结果就越可靠。

如果在测试期间生成或更改了数据，则可能需要在下一个测试周期之前恢复原始数据，以确保系统处于适当状态。

如果系统由于某些部分或数据因为任何原因无法进行性能测试，则应该使用变通方法。例如，可以使用打桩来替换和模拟负责信用卡处理的第三方组件。这个过程通常被称为“服务虚拟化”，有一些特殊工具可以协助该过程。强烈建议使用该类工具来隔离被测系统。

部署环境的方法有很多。例如，选项可能包括使用下列任何一种:

- 传统的内部（和外部）测试实验室。
- 当将系统的某些部分或全部系统部署到云上时，使用基础设施即服务(IaaS)的云作为环境
- 当供应商提供负载测试服务时，使用软件即服务(SaaS)的云作为环境

根据特定的目标和被测试的系统，一个测试环境可能优于另一个。比如：

- 为了测试性能改进(性能优化)的效果，使用一个隔离的测试环境可能是更好的选择，以便查看更改后的细小变化。
- 为确保系统能够在没有任何重大问题的情况下处理负载，要对整个生产环境进行端到端负载测试，从云或服务进行测试可能更合适。(注意，这只适用于可以通过云访问的被测系统)。
- 为了在性能测试时间受限时降低成本，在云中创建测试环境可能是一个更经济的解决方案。

无论使用哪种部署方法，都应配置硬件和软件以满足测试目标和计划。如果测试环境与生产环境匹配，则应该以同样的方式配置。但是，如果存在差异，则可能需要调整配置以适应这些差异。例如，如果测试机器的物理内存少于产品机器，则可能需要调整软件内存参数(例如 Java 堆大小)，以避免内存分页。

适当的网络配置/仿真对于全球和移动系统非常重要。对于全球系统(即，其有的用户或处理分布在世界范围的系统)方法之一可能是在用户所在的位置部署负载发生器。对于移动系统，由于可以使用不同类型的网络，网络仿真仍然是最可行的选择。有些负载测试工具有内置的网络仿真工具，也有独立的网络仿真工具。

应该正确地部署负载生成工具，并且应该配置监控工具来收集测试所需的所有度量。度量列表取决于测试目标，但是建议为所有测试收集起码的基本指标(参见 2.1.2 节)。

根据负载，特定工具/负载生成方法和机器配置，可能需要多个负载发生器。为了验证设置，还应该监控负载生成相关的机器。这将有助于避免由于其中一个负载发生器运行缓慢而无法正确维护负载的情况。

根据所使用工具设置和工具，负载测试工具需要被配置好以创建适当的负载。例如，可以设置特定的浏览器仿真参数或可以使用 IP 欺骗(模拟每个虚拟用户有不同的 IP 地址)。

在执行测试之前，必须验证环境和设置。这通常通过进行一组受控的测试并验证测试结果以及检查监控工具是否跟踪了重要信息来完成。

为了验证测试能否按设计工作，可以使用多种技术，包括日志分析和验证数据库内容。准备测试包括检查所需信息是否记录，系统是否处于正确状态等等。例如，如果测试明显地修改了系统的状态（如在数据库中添加/修改信息），则可能需要在重复测试前将系统还原到原来的状态。

4.3 执行

性能测试执行包括根据负载配置文件(通常由根据给定场景调用的性能测试脚本实现)对被测系统生成负载，监控环境的所有部分，并收集和保存所有与测试相关的结果和信息。通常高级的压力测试工具/装置自动执行这些任务（当然是在正确配置之后）。它们通常提供一个控制台在测试期间能够监控性能数据，并允许进行必要的调整(参见第 5.1 节)。但是，根据所使用的工具、被测系统和要执行的特定测试，可能需要一些手动操作。

性能测试通常关注系统稳定时的状态，即系统行为是稳定的时侯。例如当所有模拟用户/线程都启动并按预期设置执行工作时。当负载发生变化时(例如，当添加了新用户时)，系统的行为也会发生变化，从而使得监控和分析测试结果变得更加困难。达到稳定状态的阶段通常称为负载逐渐增加（ramp up），完成测试的阶段通常称为负载缓慢下降(ramp down)。

当系统的行为发生变化时，测试瞬时状态有时很重要。例如，这可能适用于同时记录大量用户或峰值测试。在测试瞬时状态时，对结果进行仔细监控和分析是非常必要的，因为一些标准方法(例如监控平均值)可能会产生误导。

在负载逐渐增加(ramp up)过程中建议实施增量负载状态，以监控持续增加负载对系统响应的影响。这确保了为负载逐渐增加(ramp up)分配足够的时间，并且系统能够处理这些负载。一旦达到稳定状态，最好监控负载和系统响应是否稳定以及随机变化（始终存在）并不大。

指定如何处理故障以确保不引入系统问题非常重要。例如，当发生故障时用户注销以保证释放系统与该用户相关的所有资源可能很重要。

如果压力测试工具中已经内嵌了监控功能并且已正确配置，那么它通常在执行测试时同时被启动。但是，如果使用独立监控工具，则应单独启动监控并收集必要信息，以便之后的分析可以与测试结果一起进行。系统日志分析也是如此。必须对所有使用的工具进行时间同步， 以便可以找到与特定测试执行周期相关的所有信息。

测试执行通常使用性能测试工具的控制台和实时日志分析来检查测试和被测系统中的问题和错误。这有助于避免不必要地继续运行大规模测试，如果出现问题，这些测试甚至可能影响其他系统（例如，如果发生故障，组件故障或生成的负载过低或过高）。这些测试的运行成本可能很高，如果测试偏离了预期的行为，则可能需要停止测试或对性能测试或系统配置进行一些即时调整。

验证直接在协议级别上进行通信的负载测试的一种技术是运行多个 GUI 级（功能）脚本，或甚至并行手动执行多个相同的操作配置文件来运行压力测试。这将检查在测试期间报告的响应时间与 GUI 手动测量的响应时间不同之处只是在客户端消耗的时间不同。

在某些情况下以自动化的方式运行性能测试时(例如，作为持续集成的一部分，如第 3.4 节所述)，由于不太可能做到实时手动监控和干预，必须自动进行检查。在这情况下，测试设置需要能够识别任何偏差或问题并发出报警（在完成正确完成测试时）。当系统的行为通常已知时，这种方法更容易进行性能回归测试，但是对于可能需要在测试期间动态调整的探索性性能测试或大规模昂贵的性能测试会比较困难。

4.4 分析结果及报告

第 4.1.2 节讨论了性能测试计划中的各个指标。预先定义这些指标决定了每次测试必须测量什么。在完成一轮测试后，应该收为定义好的指标收集数据。

在分析数据时，需先与性能测试目标比较。一旦理解了行为，就可以得出结论，该结论提供了有意义的并且包含推荐措施的总结报告。这些措施可能包含更换物理组件（如硬盘、路由器）、优化软件（如优化应用程序和数据库语句）和更改网络配置（如负载均衡、路由）。

通常分析以下数据：

- **模拟（例如：虚拟）用户的状态：**这需要首先进行检查，通常期望所有模拟用户都能够完成操作配置文件中指定的任务。此活动中的任何中断都将模仿实际用户可能遇到的情况。所以先检查所有用户活动是否都已经完成非常重要，因为遇到任何错误都可能影响其它性能数据。
- **事务响应时间。事物响应时间可以使用多种方式来测量**，包括最小值、最大值、平均值和百分位数值(如第 90 位)。其中最小和最大读数显示系统性能的极端情况。平均性能并不一定代表数学平均值以外的任何其他因素，而且经常受到其它异常值影响。第 90 个百分位值通常作为目标，因为它表示了达到特定性能阈值的大多数用户。响应时间值不建议要求达到 100%符合性能目标，因为所需的资源可能太大，而且对用户的净影响通常很小。
- **每秒事务数：**表示系统每秒可处理多少工作量(系统吞吐量)。
- **事务失效：**在分析每秒事务数值时使用该数据。事务失效表明预期的事件或流程没有完成或没有执行。有任何失效事务都需要关注和检查根因。失效的事务也可能会导致每秒事务数数据无效，因为失效的事务所消耗的时间远少于完成的事务。
- **每秒点击（或请求）数：**它模拟用户在测试期间每秒请求服务器的次数的感知。
- **网络的吞吐量：**通常以间隔时间比特数来测量，比如比特每秒。该指标表示模拟用户每秒从服务器接收的数据量。(见第 4.2.5 节)
- **HTTP 响应：HTTP 响应测量一秒内的响应数并包含可能的响应码，**比如: 200,302,404，后者表示没有找到页面。

虽然上述很多指标结果可以用表格展示，但是图形化展示更容易观察数据以及识别趋势。

用于分析数据的技术可以包括:

- 将结果与需求进行比较。
- 从结果中观察趋势
- 统计质量控制技术
- 识别错误
- 预期和实际结果比较
- 将测试结果与历史测试结果做对比

- 验证组件(如服务器、网络)是否正常运行

识别性能指标之间的关联可以帮助我们理解系统性能在什么时候开始下降。比如，当 CPU 达到 90%占用率并且系统运行缓慢时，每秒事务数是多少？

分析可以帮助了解性能下降或失败的根本原因，反过来将有利于修正。确认测试有助于确认修正措施是否解决了根本原因。

报告

分析结果与性能测试计划中的目标进行合并和比较。它们可以在整体测试状态报告中与其他测试结果一起报告，或者包含在专门的性能测试报告中。报告的详细程度应该符合干系人的需要。基于这些结果的建议通常涉及软件发布标准（包括目标环境）或者所需的性能优化。

代表性的性能测试报告包含以下方面：

总结报告

完成所有性能测试并分析和理解所有结果后，本章节即告完成。本节目标是为管理者提供简明易懂的结论、测试结果和建议，并可为后期操作提供建议。

测试结果

测试结果可能包括以下部分或全部信息:

- 一份对结果提供解释和详细说明的摘要。
- 基准测试结果作为在给定时间内系统性能的“快照”，并以此作为与后续测试比较的基础。报告结果应该包括测试起止日期/时间，并发用户目标，测度的吞吐量，以及关键发现。关键发现可能包括测量的总体错误率、响应时间和平均吞吐量。
- 一个显示任何可能(或确实)影响测试目标的高级别的组件架构图。
- 详细的测试结果分析(表格和图表)，显示响应时间、事务率、错误率和性能分析。分析还包括描述观察到的现象，例如稳定的应用程序在什么时候变得不稳定以及故障的来源(例如，web 服务器、数据库服务器)。

测试日志/信息记录

应记录每次测试执行的日志，日志通常包含以下内容：

- 测试开始的日期/时间
- 测试持续时间
- 用于测试的脚本（包含使用多个脚本的混合脚本）和相关的脚本配置数据。
- 测试使用的测试数据文件
- 测试期间创建的数据/日志文件的名称和存储位置
- 测试的硬件/软件配置信息（特别是执行之间的任何更改）
- Web 服务器和数据库服务器上的 CPU 和 RAM 的平均使用率和峰值使用率。
- 达到的性能的说明
- 识别的缺陷

建议

由测试得出的建议可包括以下内容：

- 建议进行的技术更改，如重新配置硬件或软件或网络基础设施。
- 确定需要进一步分析的领域（如分析 WEB 服务器日志可以确定帮助问题和/或错误的根本原因）。
- 对网关，服务器和网络需要进行额外监控,以便获取更详细数据来衡量性能特征和趋势（例如，性能下降）。

5. 工具– 90 分钟

关键字

负载发生器、负载管理、监控工具、性能测试工具

学习目标

5.1 工具支持

PTFL-5.1.1 (K2)了解工具如何支持性能测试

5.2 工具适用性

PTFL-5.2.1 (K4)在给定的项目场景中评估性能测试工具的适用性

5.1 工具支持

性能测试工具包括以下类型的工具来支持性能测试。

负载发生器

生成器通过一个 IDE，脚本编辑器或工具套件，能够创建和执行多个客户端实例。这些实例根据定义的操作配置文件模拟用户行为。在短时间内创建大量的实例加载在被测系统上。生成器创建负载并收集测试指标以供以后报告。

在执行性能测试时，负载发生器的目标是尽可能多地模拟真实的场景。这通常意味着需要来自不同位置的用户请求，而不仅仅是来自测试位置。使用多个存在点设置的环境将分布在负载源起的位置，这样就不会全部来自单个网络。这为测试提供了现实性，尽管如果中间网络延迟有时会扭曲结果。

负载管理控制台

负载管理控制台提供启动和停止负载发生器的控制。控制台还聚合从生成器使用的负载实例中定义的各种事务的测试指标。控制台允许查看测试执行中的报告和图表，并支持结果分析

监控工具

监控工具与被测试的组件或系统同时运行，并监督、记录和/或分析组件或系统的行为。受监控的典型组件包括 Web 服务器队列、系统内存和磁盘空间。监控工具可以有效地支持测试中系统性能下降的根本原因分析，还可以用于在产品发布时监控生产环境。在性能测试执行期间，监控器也可用于负载发生器本身。

性能测试工具的许可证模式包括具有完全所有权的传统基于席位/站点的许可证、基于云的按需付费许可证模式以及在定义的环境或通过基于云的产品免费使用的开源许可证。每种模式都意味着不同的成本结构，可能包括持续维护。很明显，对于所选的任何工具，了解该工具的工作方式（通过培训和/或自学）都需要时间和预算。

5.2 工具适用性

选择性能测试工具时应考虑以下因素：

兼容性

一般来说，选择工具是为组织的，而不仅仅是为项目的。这意味着在组织中考虑以下因素：

- 协议：如第 4.2.1 节所述，协议是性能工具选择的一个非常重要的方面。了解系统使用的协议和将要测试的协议将提供必要的信息，以便评估适当的测试工具。
- 与外部组件的接口：与软件组件或其他工具的接口可能需要被视为完整集成需求的一部分，以满足过程或其他互操作性需求（例如，在 CI 过程中的集成）。
- 平台：与组织内的平台（及其版本）的兼容性至关重要。这适用于用于承载工具的平台以及与工具交互以进行监控和/或生成负载的平台。

可扩展性

另一个需要考虑的因素是工具可以处理的并发用户模拟的总数。这将包括几个因素：

- 所需的最大许可证数量
- 负载生成工作站/服务器配置要求
- 能够从多个存在点（如分布式服务器）生成负载

可理解性

另一个需要考虑的因素是使用该工具所需的技术知识水平。这经常被忽视，并可能导致不熟练的测试人员错误地配置测试，从而提供不准确的结果。对于需要复杂场景和高度可编程性和定制的测试，团队应确保测试人员具有必要的技能、背景和培训。

监控

工具提供的监控是否足够？环境中是否有其他监控工具可用于补充该工具的监控？监控是否与定义的事务相关？必须回答所有这些问题，以确定工具是否能够提供项目所需的监控。

当监控工具是一个单独的程序/工具/整个栈时，它可以用于在产品发布时监控生产环境。

6. 参考资料

6.1 标准

- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE)软件工程-软件产品质量要求和评估

6.2 ISTQB®文档

- [ISTQB®_UT_SYL] ISTQB® Foundation Level Usability Testing Syllabus, Version 2018
- [ISTQB®_ALTA_SYL] ISTQB® Advanced Level Test Analyst Syllabus, Version 2012
- [ISTQB®_ALTTA_SYL] ISTQB® Advanced Level Technical Test Analyst Syllabus, Version2012
- [ISTQB®_ALTM_SYL] ISTQB® Advanced Level Test Manager Syllabus, Version 2012
- [ISTQB®_FL_SYL] ISTQB® Foundation Level (Core) Syllabus, Version2018
- [ISTQB®_FL_AT] ISTQB® Foundation Level Agile Tester Syllabus, Version2014
- [ISTQB®_GLOSSARY] ISTQB® Glossary of Terms used in Software Testing, http://glossary.istqb.org
-
- [ISTQB®_UT_SYL] ISTQB®基础级易用性测试大纲，第 2018 版
- [ISTQB®_ALTA_SYL] ISTQB®高级测试分析师教学大纲，2012 版
- [ISTQB®_ALTTA_SYL] ISTQB® 高级技术测试分析员大纲，2012 版
- [ISTQB®_ALTM_SYL] ISTQB®高级测试经理教学大纲，2012 版
- [ISTQB®_FL_SYL] ISTQB® 基础级（核心）教学大纲，第 2018 版
- [ISTQB®_FL_AT] ISTQB® 基础级敏捷测试大纲，第 2014 版
- [ISTQB®_GLOSSARY]ISTQB®软件测试术语表，http://glossary.istqb.org

6.3 参考书

- [Anderson01] Lorin W. Anderson, David R. Krathwohl (eds.) “A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom’s Taxonomy of Educational Objectives”,（“学习、教学和评估分类法：布鲁姆教育目标分类法的修订”）Allyn& Bacon, 2001, ISBN 978-0801319037
- [Bath14] Graham Bath, Judy McKay, “The Software Test Engineer’s Handbook（软件测试工程师手册）”,Rocky Nook, 2014, ISBN 978-1-933952-24-6
- [Molyneaux09] Ian Molyneaux, “The Art of Application Performance Testing: From Strategy to Tools（应用程序性能测试的艺术：从战略到工具）”, O’Reilly, 2009, ISBN: 9780596520663
- [Microsoft07] Microsoft Corporation, “Performance Testing Guidance for Web Applications （Web 应用程序性能测试指南）”, Microsoft, 2007, ISBN: 9780735625709

7. 索引

风险, 19, 21, 28
可扩展性测试, 9, 10
目标-问题-度量（GQM）, 16
用户思考时间, 33, 35
汇总, 16
动态测试, 11
协议, 29, 35, 42
压力测试, 9, 10
负载生成, 12, 37, 41
负载测试, 9, 10
负载配置文件, 30, 31, 32
并发性, 33
并发测试, 9, 11
吞吐量, 33, 39
批处理, 30
时间特性, 9
利益干系人, 28
利益干系人沟通, 28
系统吞吐量, 33
系统测试, 11
系统配置, 26
系统集成测试, 11
评审, 11
事务, 29
事务失效, 39
事务响应时间, 29, 39
软件即服务(SaaS), 36
质量风险, 21
服务虚拟化, 36
单元测试, 11
性能测试, 9, 10
性能测试工具, 16, 26, 41
性能测试原则, 9
性能测试脚本, 34, 37
实验, 10
组件集成测试, 11
标准 ISO 25010, 9
耐力测试, 9, 10
点击量, 39
响应时间, 14, 29
度量, 14, 16, 27
测试日志, 40
测试过程, 18
测试环境, 26, 36, 41
测试数据, 25
测量, 14
架构, 19
逐渐增加, 37
监控, 42
监控工具, 16
峰值测试, 9, 10
效率, 9
资源利用, 9
部署环境, 36
容量, 10
容量测试, 9, 11
通信协议, 29
验收标准, 22, 25
验收测试, 11
基础设施即服务(IaaS), 36
虚拟用户, 31, 33, 35, 38
常见故障, 12
综合系统, 31
缓慢下降, 37
静态测试, 11
管理控制台, 41
操作配置文件, 30, 31, 35