第5章 面向质量属性的软件体系结构设计

面向质量属性的体系结构设计是在软件体系结构设计过程中，针对软件系统所需达到的质量属性（如性能、安全性等）进行规划和决策的过程，其目的是满足利益相关者对软件系统质量的需求[16]。在软件生命周期中，面向质量属性的体系结构设计指导着系统的演化和实现，为开发人员提供了目标和指导方针。

面向质量属性的体系结构设计不仅包括对可用性、性能、安全性等关键质量属性的分析，还包括实现它们的策略。读者通过本章的学习，在掌握了面向质量属性的体系结构设计的方法和技术后，可以在实际的软件体系结构设计中，通过实施适当的策略来实现系统所需的质量属性。

5.1 软件质量属性概述

软件质量属性指软件系统的可用性、性能、安全性等可度量、可测试的属性。它们会影响到系统的运行时行为、系统设计方式以及用户的体验等方面。因此，在软件体系结构设计过程中，满足这些质量属性的需求是至关重要的。

面向质量属性的体系结构设计是在软件体系结构的设计中考虑和优化各种关键质量属性的过程，其目标是为软件系统创造一个稳定、高效和满足用户期望的环境。开发人员通过考虑不同质量属性之间的相互关系以及它们在整个系统中的影响，在软件体系结构中对其进行平衡和优化，才能构建出高质量的软件系统。

在面向质量属性的体系结构设计中，开发人员可以使用不同的策略来实现相应的质量属性，以满足软件系统的质量目标。

本章将详细讨论软件体系结构设计中的若干关键质量属性，以及实现它们的常见策略。

5.1.1 质量属性的特点

1. 质量属性属于非功能性需求

软件系统的需求可以分为功能性需求和非功能性需求。功能性需求关注软件系统应该提供哪些功能和行为，而非功能性需求关注软件系统应该如何表现和具备哪些特性。质量属性属于软件的非功能性需求，它并不描述“系统应该具备什么功能”，而是描述“系统应该如何表现和运行以达到预期效果”[17]。表5-1是某计算器App的功能性需求与非功能性需求对比。

表 5-1　计算器 App 的功能性需求与非功能性需求对比

功能性需求	非功能性需求
支持基本的四则运算	易于使用，操作简单直观
支持阶乘、平方根等特殊运算	能适应不同的手机屏幕尺寸和分辨率
支持清除、回退和重置计算	提供定制化选项，例如，调整界面布局和颜色主题
支持以特定格式显示计算结果	提供多种语言的帮助文档

2. 不同领域的软件系统关注不同的质量属性

不同的软件系统面临不同的业务领域和用户需求，往往有不同的关注重点和目标，对质量属性的要求也会有所不同。例如，一个网上银行系统可能更注重系统的安全性，以保护用户的财产不受损害；而一个手机游戏可能更注重系统的性能和易用性，以提供良好的用户体验。

在软件项目开发的初期，需要对相关的质量属性进行评估和确定，以便让后续工作更有针对性。在质量属性的确定和权衡过程中，软件开发者需要综合考虑业务需求、用户期望、可用资源和技术限制等多个因素，并与终端用户等利益相关者进行深入沟通来达到对于质量属性的共识，最终制定出适合特定项目的质量属性策略。在项目开发的过程中，也应持续地进行质量属性的评估和调整。针对持续变化的需求和风险，对系统进行相应的优化和改进。

表 5-2 是手机银行与手机游戏所关注质量属性的对比。从对比可以看出，由于手机银行和手机游戏的使用场景有很大区别，它们所关注的质量属性也有所不同。总体来说，手机银行更加注重安全性、可用性，而手机游戏则更加注重性能、易用性。

表 5-2　手机银行与手机游戏所关注质量属性的对比

手机银行关注的部分质量属性	手机游戏关注的部分质量属性
安全性（确保用户的账户和业务数据的机密性和完整性，防止未经授权的访问和欺诈活动）	性能（具备流畅的动画和图形效果，可以快速响应玩家的操作，以提供良好的游戏体验）
可用性（即使在高负载和异常情况下，也应尽量保障用户执行业务）	易用性（新手也能在较短时间内熟悉游戏的基本操作，保持较高的新手留存率）
性能（能够快速响应用户的交易请求，保证业务的处理速度和吞吐量）	可用性（避免游戏崩溃和错误发生，保护玩家的游戏进度和资料）
可修改性（为适应不断变化的金融法规和业务需求，软件能方便地进行功能更新）	可修改性（关卡、玩法等内容持续更新，以长期吸引玩家）

3. 质量属性之间可能相互抑制

质量属性之间可能存在相互抑制的关系，即在追求某一质量属性的优化时，可能会对其他质量属性产生不利的影响[18]。因此，在软件体系结构设计中，软件开发者首先需要充分了解不同质量属性之间的相互影响，之后在与软件项目的各利益相关方进行充分沟通的基础上，根据具体需求和场景在不同的质量属性之间进行权衡，以确保与不同质量属性得到妥善平衡和满足。可能的权衡方案包括确定优先级（根据系统关键需求重点关注某些质量属性）、采用折中方案（平衡不同质量属性之间的需求）和使用技术手段（利用合适的算法、模块

化设计等方式以减少质量属性之间的冲突和影响)。表 5-3 展示了某在线支付系统面临的安全性与性能之间的冲突和解决措施。

表 5-3 安全性与性能之间的冲突示例

冲 突 点	对安全性的影响	对性能的影响	解决冲突的措施
加密算法	降低被解密的可能性,提高安全性	消耗大量的计算资源和处理时间,降低性能	选择高效的加密算法和技术,或利用硬件加速和专用加速卡等技术
身份验证和访问控制	降低非法用户冒充合法用户的成功率,提高安全性	增加处理时间和计算负担,降低性能	采用异步处理和缓存技术,将一些非关键的安全和验证操作延迟处理,以减少对性能的影响
安全审计和日志记录	便于追踪和分析安全事件,提高安全性	需要完成大量的审计和日志记录操作,降低性能	通过在系统架构中引入并行处理和负载均衡技术,提高系统的处理速度和吞吐量

4. 满足质量属性应同时考虑设计、实现、部署三方面

为了确保软件系统能够真正满足质量属性的要求,仅在软件体系结构设计中考虑实现质量属性是不够的,而是需要将设计、实现和部署三个方面有机地结合起来,在不同阶段和活动中持续对质量属性进行关注。

在设计过程中:需要考虑系统架构、组件和模块的选择,以及软件模式和设计原则的应用。例如,在设计过程中,可以使用高效的算法和数据结构来提高系统的性能;采用模块化的设计可以提高系统的可修改性;引入基于角色的访问控制机制可以增强系统的安全性。

在实现过程中:需要将设计的概念转换为具体的编码实践。这需要开发人员具备良好的编码技巧和规范,以确保质量属性得到准确地实现。例如,合理地选择编程语言和框架,编写高效、可读性高且易于维护的代码,使用适当的测试技术进行代码验证等。

在部署阶段:需要考虑系统的配置、安装、集成和测试。例如,可以进行性能测试和负载测试,以验证系统的性能是否满足预期;进行系统安全审查和漏洞扫描,以保障系统的安全性。这些活动对于确保质量属性在实际运行环境中得到满足非常重要。

5. 关键的软件质量属性

本章后续将详细阐述可用性、可修改性、性能、安全性、可测试性、易用性这 6 个关键的软件质量属性,其简介如表 5-4 所示。

表 5-4 关键质量属性

名 称	简 介
可用性	软件在特定环境下持续正常运行的能力。高可用性的软件系统能够在面对异常情况时保持稳定性,并且具备快速恢复的能力
可修改性	软件系统在面对不断变化和增长的需求时,能够方便地进行修改的能力。高可修改性的软件系统能够有效地改正其中的错误,也可以灵活地增加新功能,以此适应不断变化的业务环境
性能	软件系统在执行任务时所表现出的效率和响应速度。高性能的软件系统能够在合理的时间内处理大量的请求,并且能够满足用户对实时性和吞吐量的要求
安全性	软件系统能够保护数据的机密性、完整性和可用性的能力。安全的软件系统能够防止非法访问、数据泄露和恶意攻击,并且能够及时检测和应对安全威胁

续表

名　　称	简　　介
可测试性	软件被测试的难易程度。高可测试性的软件系统应该能以较低的成本被测试，测试也较容易发现系统目前存在的缺陷
易用性	用户能够方便地使用软件系统的程度。高易用性的软件系统易于学习和操作，并能够有效地满足用户需求

此外，本章还将对特定领域关注的功耗效率、可移植性和可重用性等质量属性进行阐述。

5.1.2　质量属性场景

1. 描述质量属性的挑战

在软件系统的利益相关者描述质量属性时，通常面临层次性和隐蔽性的挑战。

层次性：质量属性通常具有层次结构，不同的属性可能相互关联。在描述时需要准确理解和识别不同层次的属性，并权衡它们之间的关系和优先级。描述质量属性常常需要利用可测量的度量标准和指标，定量化地表达其特征和要求。因此需要对数据进行收集、分析和建模。

隐蔽性：质量属性可能在设计和开发阶段并不明显，或者在实际使用中难以察觉。因此，在描述质量属性时，需要对隐含的特征和潜在问题进行预测和考虑。质量属性可能随着时间和系统演化而变化，在描述时需要考虑如何适应和应对可能的变化，以确保质量属性得到持续维护和改进。不同的质量属性可能存在冲突或平衡问题。在描述时需要平衡各个属性之间的关系，以满足系统的整体要求。

客户对软件系统质量属性的描述不清晰的示例如表 5-5 所示。

表 5-5　客户对软件系统质量属性的描述不清晰的示例

质量属性	用户的描述	用户可能遭遇的具体问题
性能	用户在使用某手机游戏时向客服反馈游戏的性能不佳	可能认为游戏画面模糊或者动画效果不流畅 可能在操作游戏时发现人物反应迟钝或者技能释放存在时间延迟 可能在游戏画面快速变化或复杂场景下出现卡顿、掉帧等情况
安全性	用户在使用某手机银行App时对客服投诉系统的安全性不佳	可能是账户被黑客等未经授权的第三方访问或被盗用 可能担心个人敏感信息（如银行卡号、身份证号等）被泄露 可能认为交易密码强度较弱，保护力度欠缺

2. 质量属性场景

为了改善用户描述不具体导致的对质量属性理解不清晰，软件工程研究者引入了“质量属性场景”这一工具[19]，用以清晰地描述软件系统对质量属性的要求。一个质量属性场景包含以下 6 个组成部分。

（1）刺激源。

在质量属性场景中，“刺激源”指可以触发系统做出响应的外部事件或条件。表 5-6 是一些常见的刺激源。

表 5-6 质量属性场景中常见的刺激源

刺激源形式	具体描述
用户请求	用户在系统界面上进行的操作，如单击按钮、填写表单、发送请求等
数据输入	系统接收到的数据输入，可能是来自用户、外部系统或其他数据源的信息
系统负载	系统面临的工作负荷，包括并发请求数、数据量等
时间	系统需要根据不同的时间段或时序对事件做出不同的处理
特定事件	特定的外部事件触发系统的反应（如系统故障、异常情况、定时任务等）
外部影响	外部因素对系统的影响（如网络状况、环境条件、外部系统的状态等）

（2）刺激。

在质量属性场景中，“刺激”是指刺激源对系统的具体影响或要求，即刺激源引发的系统行为或变化。刺激描述了外部事件或条件所带来的具体需求或影响，帮助开发者了解系统需要做出何种反应。表 5-7 是一些常见的刺激。

表 5-7 质量属性场景中常见的刺激

刺激形式	具体描述
请求响应时间	要求系统在接收到用户请求后，在特定的时间内进行响应
并发请求量	要求系统能够同时处理多个并发的用户请求，而不会出现性能问题
数据输入规模	要求系统能够处理不同规模的数据输入
实时性要求	要求系统能够实时处理数据或提供实时的响应，以满足特定业务需求
可用性要求	要求系统在规定时间内保持高可用状态，以确保用户可以随时访问系统
安全性要求	要求系统在处理用户请求和敏感数据时采取一定的安全措施，以保护数据安全
扩展性要求	要求系统在面临增加负载或用户数量时能够水平扩展，以保持稳定性和性能

（3）制品。

在质量属性场景中，“制品”是指系统中被刺激所影响的部分。表 5-8 是一些常见的制品。

表 5-8 质量属性场景中常见的制品

制品形式	具体描述
系统整体	与性能等质量属性相关
系统对外提供服务的能力	与可用性、安全性等质量属性相关
系统的代码	与可修改性、可测试性等质量属性相关
系统的数据	与安全性等质量属性相关

（4）环境。

在质量属性场景中，“环境”是指在刺激发生时软件系统所处的状态。在刺激发生时，系统可能处于正常状态，也可能处于超负荷运行下的过载状态甚至已经处于由轻微故障所导

致的降级状态。

（5）响应。

在质量属性场景中，“响应”是指软件系统对于特定刺激所做出的具体反应或行为。它描述了系统如何处理请求、生成结果或执行操作，以满足特定的需求或要求。表 5-9 是一些常见的响应。

表 5-9　质量属性场景中常见的响应

响应形式	具体描述
请求处理	系统接收到用户请求后的处理方式，包括数据处理、计算、查询数据库等操作
结果生成	系统在处理请求后生成的结果或输出，包括信息、报告、图形等形式
操作执行	系统在接收到指令或触发事件后的具体操作，可以是更新数据库、调用外部服务、发送通知等
错误处理	系统在出现错误或异常情况时的处理方式，如错误提示、数据回滚、日志记录等
系统状态变更	系统在响应过程中可能发生的状态变更，如更新数据、保存状态等

（6）响应度量。

在质量属性场景中，“响应度量”是指对系统响应进行量化度量的指标或方法。通过分析响应度量，可以评估系统是否满足要求，并采取相应的优化措施。表 5-10 是一些常见的响应度量。

表 5-10　质量属性场景中常见的响应度量

响应度量形式	具体描述
响应时间	衡量系统在用户请求发出后返回结果所需要的时间
吞吐量	衡量系统单位时间内能够处理的请求数量
错误率	衡量系统在处理请求过程中产生的错误或异常情况的比例
安全指标	衡量系统在保护用户数据和防范安全威胁方面的表现，如认证成功率等
规模适应性	衡量系统在面对不同规模和负载条件下的性能表现能力

综上所述，在质量属性场景中，刺激源是触发系统响应的起点，刺激描述了具体的要求或影响，制品指出了系统受到刺激所影响的部分，环境定义了系统操作所处状态，响应描述了系统对刺激做出的反应，响应度量衡量了系统响应的质量。这 6 个部分共同构成了完整的质量属性场景，帮助开发者理解系统的质量属性要求。表 5-11 以“汽车在行驶中加速”的质量属性场景为例，说明质量属性场景的 6 个组成部分。

表 5-11　“汽车在行驶中加速”的质量属性场景

组成部分	具体描述
刺激源	触发汽车加速的源头，可以是驾驶员踩下加速踏板、自动驾驶系统产生的指令或传感器检测到的行驶条件等
刺激	对汽车的加速请求，可以是要求汽车加快速度的指令

续表

组成部分	具体描述
制品	在加速场景中，制品包括发动机、传动系统、加速踏板、驱动控制单元等汽车加速涉及的软件、硬件
环境	当前车辆可能处于正常状态，也可能已经处于涡轮增压的状态
响应	汽车加速行为的实际表现，包括引擎增加功率、传动系统调整齿轮比等
响应度量	对汽车加速过程进行量化评估的指标，包括加速时间、加速度、燃油消耗等

5.2 可用性

5.2.1 可用性的含义

1. 可用性的概念

可用性是指软件系统能够在要求的时间内被正常使用的能力，强调了软件系统在面对各种故障时的稳定性和可靠性。对于支撑关键业务功能的软件系统而言，可用性是至关重要的质量属性。

软件可用性的目标是确保软件系统在用户需要时可以被正常使用，并尽可能地减少由于故障、错误或其他异常情况引起的系统工作状态的中断。表 5-12 列出了在软件体系结构设计中可用性需要考虑的方面。

表 5-12 可用性需要考虑的方面

方面	详细描述
可靠性和容错性	具有高可靠性和容错性的架构，以保证系统部件（如服务器、数据库、网络等）的连续运行，并能够在发生故障时在较短时间内恢复正常
冗余和备份	引入冗余和备份（如使用备用服务器、故障转移系统等），以确保即使出现故障，系统仍然能够正常运行
错误处理和恢复机制	具备有效的错误处理和恢复机制，能够捕获和处理异常情况，并采取适当的措施进行错误修复和数据恢复，以保证系统的连续可用性
监控和管理	对软件系统的健康状态进行实时的监控和管理，以便及时检测和响应故障、错误等异常情况

需要注意的是，提前确定的停机维护并不被计入不可用的时间。例如，网络游戏的运营商在修复游戏 bug 或更新游戏版本等情况下，需要在特定时间段进行系统维护。此时，停机时间被事先通知给玩家，并不被计入游戏的不可用时间。通过这种方式，游戏运营商可以利用停机时间完成系统维护、数据库清理、应用安全补丁、更新功能等操作，以提高游戏环境的质量和用户体验。虽然在维护期间，玩家不能访问游戏，但是这种提前确定的停机维护不会对游戏的整体可用性产生负面影响。相反，对于游戏运营商来说，合理计划并提前通告的停机维护有助于减少故障对玩家正常游戏的影响，可以提高玩家对游戏可用性的信任。

2. 可用性的关注点

在关注软件系统的可用性时，系统的利益相关者不仅需要考虑到系统是否发生故障，还

需要对故障的后果进行评估和分析。

关注系统中是否发生了故障：故障可能是由于硬件故障、网络中断、软件错误、非预期的用户操作或其他不可预见的原因引起的系统中断或异常。在软件体系结构设计中，利益相关者应该设定监控和警报机制，以便及时检测和识别系统中的故障情况。

评估故障的后果：故障可能会对系统的功能、性能、安全性等方面产生不同程度的影响。从可用性的角度来看，利益相关者主要关注故障对用户体验和用户满意度的影响。例如，系统的故障可能导致用户无法访问关键功能、数据丢失、任务无法完成、操作变得缓慢或不可预测等。

故障对软件系统涉及的不同利益相关者所造成的影响也并不相同，具体如表 5-13 所示。

表 5-13　故障对软件系统不同的利益相关者所造成的影响

影响方面	详细描述
用户体验和满意度	故障可能导致用户无法完成关键任务、造成数据丢失、降低操作效率等，从而影响用户体验和满意度
业务连续性	对于企业和组织，软件系统的可用性直接影响业务的连续性。故障可能导致业务中断、订单延误、客户流失、营收损失等
系统运维和支持	故障会对系统运维和支持团队产生额外的工作负担。团队需要花费时间和资源来处理故障、恢复系统和提供技术支持
品牌声誉和信任度	软件系统的可用性与企业或组织的品牌声誉和信任度直接相关。如果系统频繁故障或长时间不可用，用户会对企业的服务质量产生怀疑，导致企业品牌声誉受损

3. 可用性的度量指标

为了更加全面地度量软件系统的可用性，可以使用度量系统可靠性、可恢复性、可用性概率三个方面的指标，详细说明如表 5-14 所示。

表 5-14　可用性的度量指标

类　型	具体指标	目　的
度量系统可靠性	平均无故障时间、平均故障间隔时间	度量系统的稳定性和故障概率
度量系统可恢复性	平均恢复时间、恢复点目标、恢复时间目标	度量系统从故障中恢复的速度和数据恢复的准确性
度量可用性概率	可用性百分比	直观地表示系统可用性的程度

不同的可用性度量指标适用于不同类型的应用和系统，因此在评估可用性时需要根据具体情况选择和使用合适的指标。可用性度量指标的重要性在不同应用场景下也存在很大差异。对于实时系统或关键业务系统，高可用性是很重要的。而对于非关键系统，低成本的可用性方案可能更为合适。

下面以阿里云为例，说明国内软件平台的高可用性设计。阿里云是国内领先的云计算服务提供商，高可用性是其占据市场份额的关键之一。阿里云的高可用性架构设计和实践经验不仅展示了中国科技企业在创新方面的能力，也有力推动了数字经济的发展和企业升级。为了实现高可用性，阿里云所提供的特性如表 5-15 所示。

表 5-15 阿里云为实现高可用性所提供的特性

特 性	说 明
架构设计	采用分布式架构设计，利用多个数据中心和服务器集群来实现高可用性；使用硬件设备冗余、数据冗余、网络冗余等多重冗余机制来防止单点故障
快速故障恢复	采用自动化的监控和运维技术，快速检测到故障并进行响应，实现分钟级别的故障恢复时间
数据保护和一致性	采用强大的备份和复制技术，确保用户数据可以快速备份、恢复和同步；提供可靠的数据存储解决方案，保证数据的一致性和完整性
性能和弹性扩展	具有高性能和弹性扩展的能力。采用分布式计算和存储技术，能够处理大规模的并发请求和海量的数据。提供弹性资源调度机制，可以根据需要快速扩展或缩减资源，以满足用户的需求

5.2.2 可用性的质量属性场景

1. 可用性质量属性场景的 6 个组成部分

可用性质量属性场景的 6 个组成部分的常见情况如表 5-16 所示。

表 5-16 可用性质量属性场景的 6 个组成部分

组 成 部 分	具 体 描 述
刺激源	引发系统响应的外部因素，包括用户请求、网络传输问题、硬件故障等
刺激	用户请求刺激包括用户发送请求、输入数据等方面。网络传输问题刺激包括网络延迟、丢包等方面
制品	数据库、计算资源、网络等
环境	系统可能处于正常运行的状态，也可能是已经处于濒临瘫痪的“亚健康”状态等
响应	记录错误报告并回传给厂家、通知管理员或其他系统、临时关闭系统等
响应度量	故障时间百分比、故障恢复时间、平均无故障时间等

为便于读者理解 6 个组成部分中的环境和响应，下面举例说明。

环境： Windows 系统在日常使用中可能会因为恶意软件、驱动程序故障等问题而无法正常启动。为解决这些问题，Windows 系统提供了“安全模式”功能。安全模式允许用户在系统启动时加载最少的设备驱动程序和系统服务，以便用户能够诊断和解决系统启动问题，其特点如表 5-17 所示。

表 5-17 Windows“安全模式”的特点

特 点	具 体 描 述
可靠性和稳定性	通过最小化加载系统组件和驱动程序，降低了系统在启动过程中出现问题的可能性，提高了系统的可靠性和稳定性
可恢复性和容错性	当系统遇到启动问题时，用户可以进入安全模式进行故障诊断和修复，而不至于导致系统完全无法使用；帮助用户迅速恢复系统功能，降低了系统故障对用户造成的影响
适应性和灵活性	用户可以根据实际情况选择不同的安全模式选项（如带网络支持的安全模式、带命令提示符的安全模式等），以满足不同故障情况下的需求

响应：软件在使用过程中难免会发生错误和异常情况，为了改善软件的可用性，许多软件在设计中引入了错误报告机制。软件会在遇到错误时自动发送错误报告给开发者，从而帮助开发者更好地理解和解决用户面临的问题。图 5-1 展示了当 Word 发生错误时弹出对话框告知用户并询问是否愿意发送错误报告。通过收集和分析错误报告，开发者可以深入了解软件在实际环境中的问题和异常情况，快速定位和修复潜在的缺陷和漏洞。

图 5-1　Word 发生错误时弹出对话框告知用户

2. 可用性质量属性场景示例：系统收到来自外部的异常信息

某软件系统在运行时由于故障收到了异常信息，此时的可用性质量属性场景如表 5-18 所示。

表 5-18　可用性质量属性场景示例（系统收到来自外部的异常信息）

组成部分	具体描述
刺激源	使用系统的用户
刺激	用户向系统发送消息时，由于网络异常，导致系统收到的消息是无法识别的
制品	影响系统接收和处理消息的相关模块或组件
环境	当前系统处于正常的状态
响应	系统发现消息无法识别后，丢弃了消息并给用户发送提醒
响应度量	系统在 0.1s 内完成了消息的处理，并未影响系统的正常运行

5.2.3　可用性的实现策略

策略是用于满足特定质量属性的具体措施。例如，对于提高系统的可用性，可能会采取容错设计、负载均衡、监控和报警等策略。策略是软件体系结构中体系结构风格的基本单元，不同的体系结构风格包含不同的策略集合，以实现相应的质量属性。例如，层次体系结构风格强调模块化和分离关注点，通过将系统划分为多个层次进行组织，以实现可修改性、可测试性等质量属性。

对于软件系统，可用性的目标主要是降低故障对系统正常运行和用户体验造成的影响。故障可能来源于硬件故障、软件错误、网络问题等。通过采取适当的策略，利益相关者可以减少故障导致的服务中断和数据丢失等不良影响，使系统能够持续地提供稳定和可靠的